CS 106
A. I. 65

A PROGRAM TO PLAY CHESS END GAMES

BY

BARBARA J. HUBERMAN

SPONSORED BY
ADVANCED RESEARCH PROJECTS AGENCY
ARPA ORDER NO. 457

TECHNICAL REPORT NO. CS 106
AUGUST 19, 1968

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO AI-65

August 19, 1968

CS 106

A PROGRAM TO PLAY CHESS END GAMES

by Barbara Jane Hubermar

ABSTRACT: A program to play chess end games is described. The model used in the program is very close to the model assumed in chess books. Embedded in the model are two predicates, better and worse, which contain the heuristics of play, different for each end game. The definitions of better and worse were obtained by programmer translation from the chess books.

The program model is shown to be a good one for chess end games by the success achieved for three end games. Also the model enables us to prove that the program can reach checkmate from any starting position. Insights about translation from book problem solving methods into computer program heuristics are discussed; they are obtained by comparing the chess book methods with the definitions of better and worse, and by considering the difficulty encountered by the programmer when doing the translation.

The research reported here was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-183).

ACKNOWLEDGEMENTS

I would like to express deepest thanks to my thesis advisor, Professor John McCarthy, for his many valuable suggestions and helpful criticisms. Also I am grateful to Professor J. Feldman for his constructive reading of the final version of the thesis, and to Professor R. Reddy for his earlier reading and assistance.

In addition I am indebted to my colleague, Mr. John Lennie, for his critical evaluation of parts of this work, and to my cousin, Mrs. Jill Custer, for her encouragement and careful reading.

I wish to express my appreciation to Mrs. Judy Muller for her excellent typing and preparation of this report, and to Mrs. Dorothy McGrath for her fine illustrations.

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense, (SD-183).

TABLE OF CONTENTS

Table of Contents (cont'd.)

FIGURES

Figures (cont'd)

CHAPTER 1

INTRODUCTION

This research is concerned with the process of translating book descriptions of problem solving methods into program heuristics. Many books have been written for the purpose of teaching how to perform some task. The task under discussion may be almost any kind of activity, including intellectual activities such as proving theorems in geometry or solving differential equations. People are able to learn from these books although the difficulty in learning varies from task to task. Therefore we can consider the information in the books as sufficient for people. It would be convenient if the book information could be used by computer programs. We are interested in whether the information is sufficient for computers, and if not, then we want to know what kind of additional information is needed.

The fact that book information is sufficient for people does not mean that it can be used directly. If the book describes an algorithm, then sometimes only memorization is required of the reader; for example, the method of finding truth values of sentences in propositional calculus by means of truth tables can be learned by memorization. Many tasks, however, require substantial learning before the student can understand the book. The task of playing chess end games by computer provides a simple but not trivial area for this research. By chess end games we mean those games where the number of pieces on the board is small, but

the number of moves to checkmate large: for example, Two Bishops and King against King, or the various Pawn endings. Chess books give rules for these end games which are not algorithms but are supposed to be simple and complete enough that beginners at chess can learn to play the end games fairly easily. A certain amount of intelligence is required of the student, but still we expect to need only a minimal amount of additional information. In this study the programmer will do the translation. Since this translation from the chess books to the program is not direct, as it would be in the case of truth tables, we expect to learn something from the translation process.

Methods and Models

Computer researchers are well aware by now of the fact that any task requiring intelligence can be profitably approached by distinguishing between models and methods. The model, which is a representation of the structure of the problem [Minsky, 1961][1], determines the overall logic of the program. The methods are the heuristics which the program uses within this structure. For example, in the Logic Theory Machine [Newell, Shaw, and Simon, 1957], the model is a backwards tree and is represented by that part of the program called the "Executive Routine". Within this framework substitution, detachment and chaining methods are used; these are encodings of the way people apply the rules of inference in propositional calculus.

Generally books are concerned only with teaching the methods which should be used to solve problems in the task area. The methods must be

1. See page 413 of Minsky [1961].

applied within a structure which is assumed in the book but not generally defined explicitly. It is necessary to build a model of this structure in the computer before information about methods can be taken from the book.

We expect that different models are required for different tasks. Very often the model is a backwards tree; the General Problem Solver [Newell and Simon, 1961] is based upon this fact. However there are problems which would require a different model: for example, bidding in bridge. The closer the model used in the program is to the way that the author of the book thinks about the problem, the easier it will be to translate the methods of the book into heuristics for the program. Chess end games could be handled by the General Problem Solver; however in this research a model is used which is much closer to the abstract model assumed in the chess books. In this way we hope to eliminate making changes in the methods to account for a difference between the program's model and the abstract model assumed in the book. This means that any difficulty experience in translating the book methods into program heuristics can only be due to inadequacy in the method descriptions.

Model and Methods for Chess End Games

The model used for chess end games is a forcing tree. The program is supplied with two functions better and worse (containing the methods) which compare positions. From a given starting position p , in which the program has the move, it uses tree search to find positions q which are better than p . It will search until such a position q

is found for every sequence of moves by the opposition. An example of such a tree is given in Figure 1.1. The program will then make the moves dictated by the tree until it reaches a q at the end of a branch in the tree; then it recalculates the tree to force positions better than q . This process continues until checkmate is reached. worse is used by the program to cut off branches of the tree which lead to disaster (stalemate, etc.), and also to prune the tree. This model is described in detail in Chapter 2.

The forcing tree model will be used for all the different end games. However each end game is played by different methods which will result in different definitions of better and worse. This enables us to examine the problems of translation from methods to program heuristics several times and for games of varying degrees of difficulty.

better and worse are built up out of pattern recognition functions of positions which can be defined in a natural manner from information given in the chess books. The methods, or rules, of play are defined in two ways in the books. First of all, written statements are made. For example, in the description of the Rook and King against King game in Capablanca [1935] we find: "The principle is to drive the opposing King to the last line on any side of the board" and then the student should "Keep his King as much as possible on the same rank, or...file, as the opposing King".[2] The play of other games (and in other books) is described by similar rules. It is not difficult to convert a principle into a pattern recognition function of positions because the pattern is inherent in the principle. For example, to express the

2. See pages 26 and 27 in Capablanca [1935]

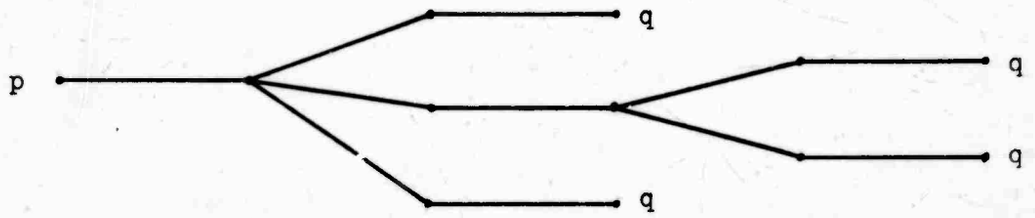


Figure 1.1. Example of a Forcing tree. The program has the move in p; it must make a move leading to a position q judged <u>better</u> than p for every sequence of moves by the opposition. Each iteration of the program will produce a tree like this; several iterations will be required to reach checkmate.

first principle quoted above we define

$f(x) \equiv$ the opposition king is confined to an edge of the board in x,

for x a position. Then we might decide a position q was better than position p if

$$f(q) \wedge \neg f(p)$$

because the principle is satisfied by making the moves leading from p to q.

The chess books supplement the principles with examples of program play. The principles generally cover the gross features of the game and form a framework for viewing the play of the game. The majority of moves are only partly derived from the principles; they are more directly derived from the examples of program play. The examples contain more or less complete information about methods of play; the difficulty comes in deciding what pattern features of the positions are important. Obviously, induction is required to make this decision. Each example is considered representative of a large class of positions and a general rule must be defined for that class. If the example is accompanied by principles, this simplifies the induction by providing clues to important features (see Figure 3.1). The induction leads automatically to the kind of pattern recognition functions used in better and worse.

Goals of the Research

The primary goal of the research is to study the translation process. We begin by stating two criteria which will help us achieve this goal. First we would like to see if our model is a good one for chess end games. Our first hypothesis is: the model used in the

program is a good representation of the abstract model assumed by chess books. We can support this hypothesis by successfully running the program on different end games. Furthermore, conditions can be given on better and worse which permit us to prove informally that the program works correctly. The proof depends heavily on the model and could not be given for a different model (for example the General Problem Solver model).

Our second hypothesis is: the information in the chess books is sufficient for the definitions of better and worse. The chess book information will suffice for worse if all disastrous positions are described. For better much more information is needed; the books must give rules for recognizing progress frequently enough that the tree search between positions is reasonable. For example it is not enough to have rules recognizing only checkmate positions.

Finally we turn our attention to the primary goal of studying the translation process. We assume that the two criteria are satisfied. First we consider how closely the definitions of better and worse correspond to the chess book methods, measuring the correspondence by comparing program play with the book examples. Also we consider the difficulty encountered in defining better and worse.

Outline of the Thesis

In Chapter 2, the overall organization will be described. A detailed definition of the uses of better, worse, and tree search will

be given; this constitutes the model which we use for chess end games.

In Chapter 3 the form of the contents of functions better and worse will be discussed. These functions are different for each end game, since different methods are used for each game. However, the form given for better and worse is used in all end games. Some rules are given for better and worse which will enable us to prove that the program is correct in the sense of being able to achieve checkmate from a given starting position.

Chapters 4, 5, and 6 each describe the definitions of better and worse for a different end game. Rook and King against King is discussed in Chapter 4, two Bishops and King against King in Chapter 5, and Bishop, Knight and King against King in Chapter 6. These games are presented in order of difficulty. The rook end game is quite a simple one; two Bishops is a game of moderate difficulty, while the Bishop-Knight end game is very difficult. The process of translating from the book information into pattern recognition functions will be described, and reasons will be given for the programming decisions. Examples of program play will be included for each game.

Chapter 7 contains an informal proof of program correctness. This proof is given after the various end games are described because it depends on the heuristics used for each game.

Chapter 8 will contain an evaluation of the better, worse format in terms of the two primary goals. Subjects covered will include program efficiency, a description of a way to have the program do some of the inductive learning, and extensions to other task areas.

In the following chapters, ordinary chess notations will be used [Capablanca, 1935]. The program is written in LISP [McCarthy, Abrams, Edwards, Hart and Levin, 1965], and the reader is expected to have some knowledge of this language. Function definitions are given using notation and basic functions which are defined in Appendix A. They are built up of the connectives $\equiv$ (equivalence), $\supset$ (implication), $\wedge$ (conjuction), $\vee$ (disjunction), and $\neg$ (negation). These are used in the same way LISP (not ALGOL) uses them; i.e., if in $p \wedge q$, p is evaluated and found to be false, then q is not evaluated.

BLANK PAGE

CHAPTER 2

PROGRAM ORGANIZATION

Notation

Throughout this thesis, certain conventions of notation will be used. As in the ordinary use in chess books, the white side is the winning side. The program will play white and a person black. The letter p, with possibly subscripts or superscripts, is used to represent a position with white (program) to move, and q, again with subscripts or superscripts, for positions with black to move. When the color of the move is unimportant, letters x, y, etc., with subscripts or superscripts will be used.

In a position p, a certain set of white moves is legal according to the rules of chess. A legal move is made from p to produce a new position q with black to move. We will represent the connection between p and q by means of the relation M_W. The statement pM_Wq is read: q is a position which results from making one legal white move in p. Similarly we write qM_Bp which means p is a position which results from making one legal black move in q. If pM_Wq we say q is an immediate successor position of p, and similarly for qM_Bp. If we say that q is an ultimate successor of p this means there exist $p_1,\ldots, p_n$ and $q_1,\ldots, q_n$ such that

$$pM_Wq_1 \wedge q_1M_Bp_1 \wedge \ldots \wedge q_nM_Bp_n \wedge p_nM_Wq .$$

The program is given as a starting position a position p with white to move. In some end games, white can win only from certain legal positions with white to move. Let

$P = \{p \mid p$ is a legal position with white to move, and white can win from $p\}$.

The program must work correctly for any starting position $p \in P$; we do not care what happens for $p \notin P$.

As explained in Chapter 1, better and worse are used to compare positions. They both have as an argument a pair of positions (p, q) . The first position is always a position with white to move; the second is always a position with black to move. q is either an immediate or ultimate successor to p .

The statement better(p, q) is (not) true is equivalent to saying q is (not) better than p , and similarly worse(p, q) is (not) true is equivalent to q is (not) worse than p . Occasionally when discussing a tree search a statement like " q is a better position" will be made. This means q is better than the p at the head of the tree. better and worse will always be underlined; so will all other function names except those consisting of only one letter.

Program Organization

To start with, the program is given an initial position $p \in P$. It generates all positions q such that $pM_w q$. The order in which these positions are generated is not important; let us refer to them

as $Q = \{q_1, \ldots, q_n\}$. For each q_i the program asks the question worse(p, q_i) . If q_i is worse than p then q_i is immediately rejected by the program. If worse(p, q_i) is false, then the program asks better(p, q_i) . If better(p, q_i) is true, the move which led to q_i is retrieved by the program and made at this point without any further analysis or examination of the remaining positions $q_{i+1}, \ldots, q_n$. Figure 2.1 is a flowchart of this part of the program.

If all q_i have been examined and none is found which is both better and not worse than p , the program will resort to tree search. The work it has done so far is really the first level of the tree search. A branch remains in the tree for each q_i which was not worse than p . Call this set Q_1 .

During the tree search the first element of the argument pair of better and worse remains the initial position p . As explained previously, the second element must be a position with black to move. This means that in the tree search, the ends of the branches can't be evaluated after every move, since half of the moves result in positions with white to move. Also it is convenient to have the depth in the tree equal to the number of white moves required to get to that point. If a position q is said to be at depth n in the tree, this means that $2n-1$ moves are required to get to q ; of these n are white moves and $n-1$ are black moves.

The basic premise of this method of play is that from p white is able to force a position q better than p . "Force" means that white must be able to answer every black move with an eventual better position;

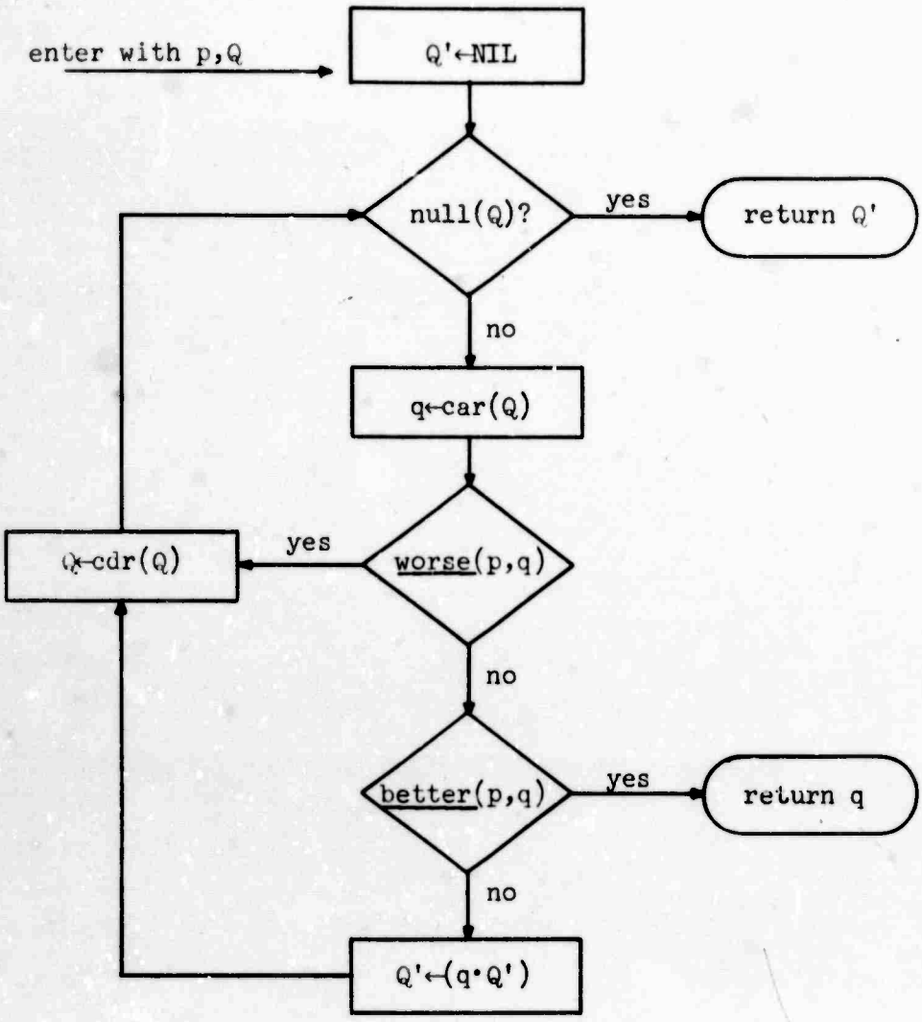


Figure 2.1. BW(p,Q)

p ∈ P is a starting position
Q is a list of successor positions of p .

BW returns
a single position q ; this means q is better and not worse than p
a list of positions (possibly empty) containing all positions which were not worse than p ; this means no member of Q is better and not worse than p .

conversely if any black move results in all positions worse than p, the position in which that black move was made must be discarded.

The tree search is a breadth-first search. For each q_i, the program generates $P_i = \{p_{i1}, \ldots, p_{is_i}\}$. Each p_{ij} is the result of a legal black move in q_i; i.e., $q_i M_B p_{ij}$ for $j = 1, \ldots, s_i$. Then for each $p_{ij} \in P_i$ the program generates $Q_{ij} = \{q_{ij1}, \ldots, q_{ijs_{ij}}\}$ where $p_{ij} M_W q_{ijk}$ for $k = 1, \ldots, s_{ij}$. The program then computes $BW(p, Q_{ij})$ (see Figure 2.1); that is, the q_{ijk} are compared with p in the same way in which the q_i were compared with p previously. In order for the move leading to q_i to be accepted by the program, for each p_{ij} there must exist a q_{ijk} such that worse(p, q_{ijk}) is false and better(p, q_{ijk}) is true; that is, $BW(p, Q_{ij})$ must return a single position for $j = 1, \ldots, s_i$ (i.e., for every black move q_{ij}). If this happens, then the move leading to q_i is made by the program without examining the other $q_i \in Q_1$.

If $BW(p, Q_{ij})$ returns the null list for some Q_{ij}, this means that all $q_{ijk} \in Q_{ij}$ are worse than p. This happens because in q_i the black move leading to p_{ij} is permitted, and white is not in a position to control the result. In this case q_i is completely removed from the tree, just as if it had been worse than p in the first place. The move q_2 is eliminated in this way in Figure 2.2.

If q_i is neither rejected nor accepted, then for one or more of the p_{ij}, there exist several q_{ijk} such that worse(p, q_{ijk}) is false but better(p, q_{ijk}) is also false. In this case, $BW(p, Q_{ij})$ returns the list of such q_{ijk}; this information is saved in Q_{ij} in

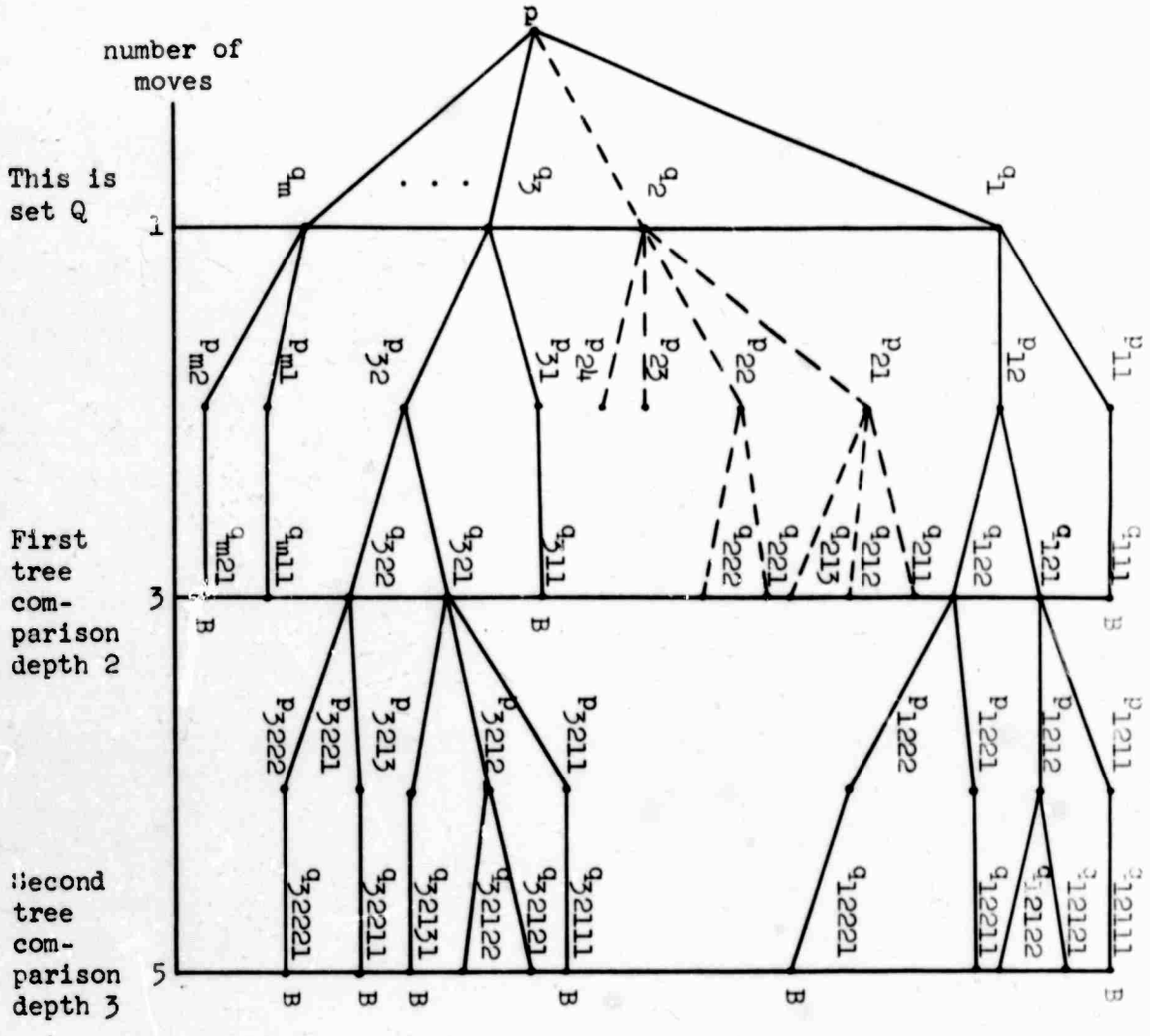


Figure 2.2. Example of Forcing Tree.

1. From position q_2, for the black move leading to p_{23}, all white moves lead to positions worse than p. Therefore this branch is eliminated. p_{24} will not be examined.
2. Positions which are better than p are marked with a B. A branch is accepted when every termination is marked B. Note that even if a single position with white to move remains at a level, it is not necessarily better; e.g., q_{m11}. This would be true even on the very first level (set Q).
3. No decision is made at depth 2.
4. Now depth 3 is begun. For q_1 no decision is made and all information is saved.
5. The branch for q_3 is examined next, and it is accepted since the end of every branch is marked with a B. One branch ends at depth 2; the others end at depth 3. The program will now make the move leading to q_3. It does not examine the remaining branches for $q_4, \ldots, q_m$.

case no q_i is accepted at this level. For example in Figure 2.2, we would have set $Q_{12} = \{q_{121}, q_{122}\}$.

If no q_i is accepted by the program at this level, the program extends the tree one more level every place where a decision wasn't made previously (where a list Q_{ij} is saved). Every element $q_{ijk} \in Q_{ij}$ produces several lists of positions Q_{ijkm} , one for each immediate successor position p_{ijkm} to q_{ijk} . Now $BW(p, Q_{ijkm})$ is called. If it returns a single position for each immediate successor p_{ijkm} of q_{ijk} , then q_{ijk} is accepted at depth 2 (just as before q_i would have been accepted at depth 1). In this case the other members of Q_{ij} are not considered. Also, as before, a branch can be rejected, either back to depth 2 (q_{ijk}) or all the way back to depth 1 (q_i) .

If no decision is made at depth 3, the program goes down another level to depth 4. The search is continued until a decision is made. Figure 2.2 is an example of a position which required a search of depth 3. No decision was made for q_1 at depth 3 so all the information in the figure would have been saved. For q_3 , only one black move p_{32} remained to be answered and q_{322} is accepted at this level. Therefore q_3 is accepted by the program at this point, and $q_4, \ldots, q_m$ are not examined.

When the program has selected a branch of the tree, it remembers the tree, and will make the moves dictated by the tree for as long as it lasts. This is a very important point since it is the feature which enables the program to force a <u>better</u> position.

Figures 2.3 to 2.6 are flow charts of the program. Figure 2.3 is the main program; the other three flow charts cover the tree search.

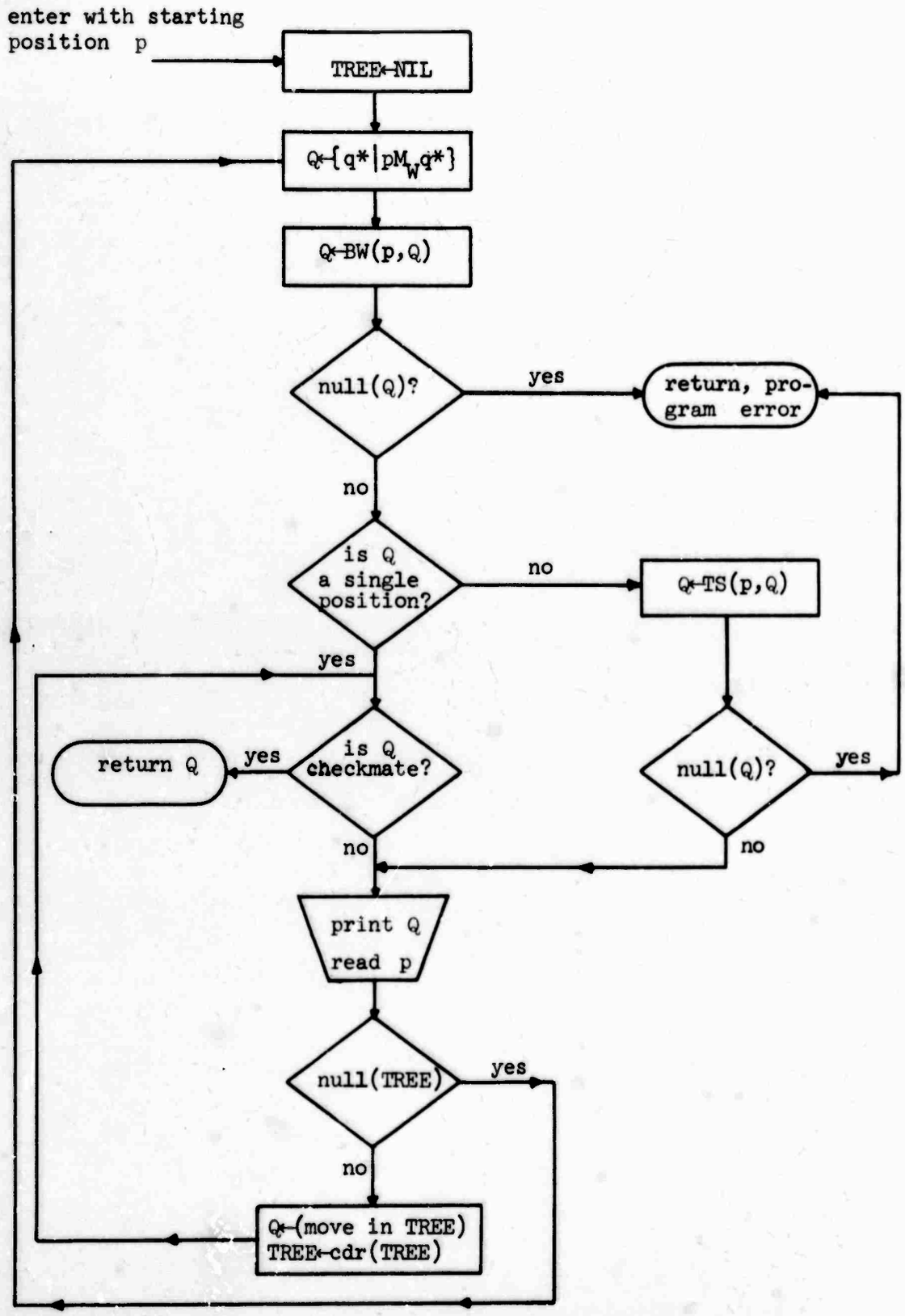


Figure 2.3. Main Program Flow.

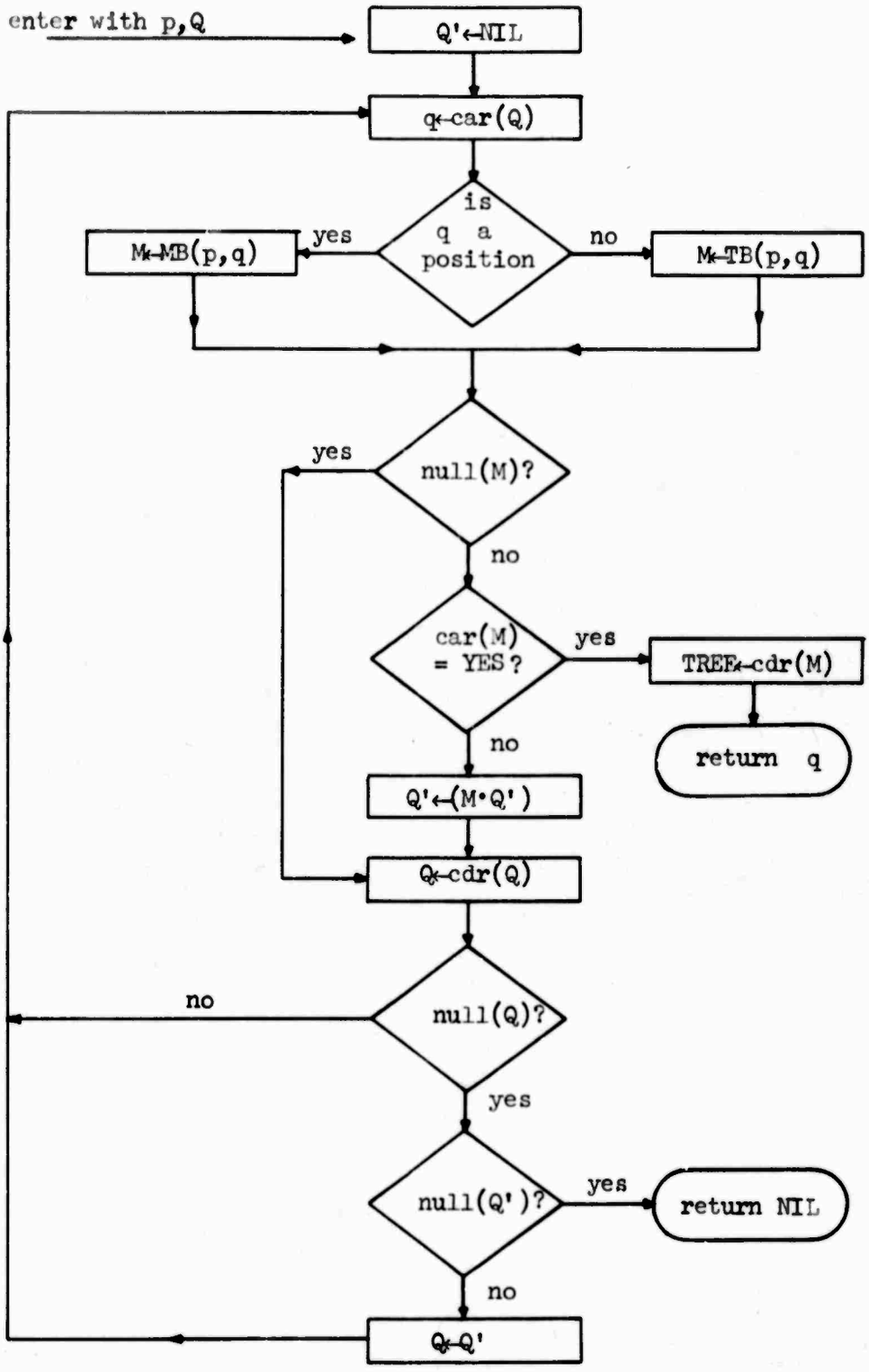


Figure 2.4. Function TS(p,Q) is the top level tree search function which starts the tree search going; calls the functions which follow the branches of the tree; returns the selected position and saves the branch in TREE if a decision is made; or starts again to extend the search one more level if no decision is made.

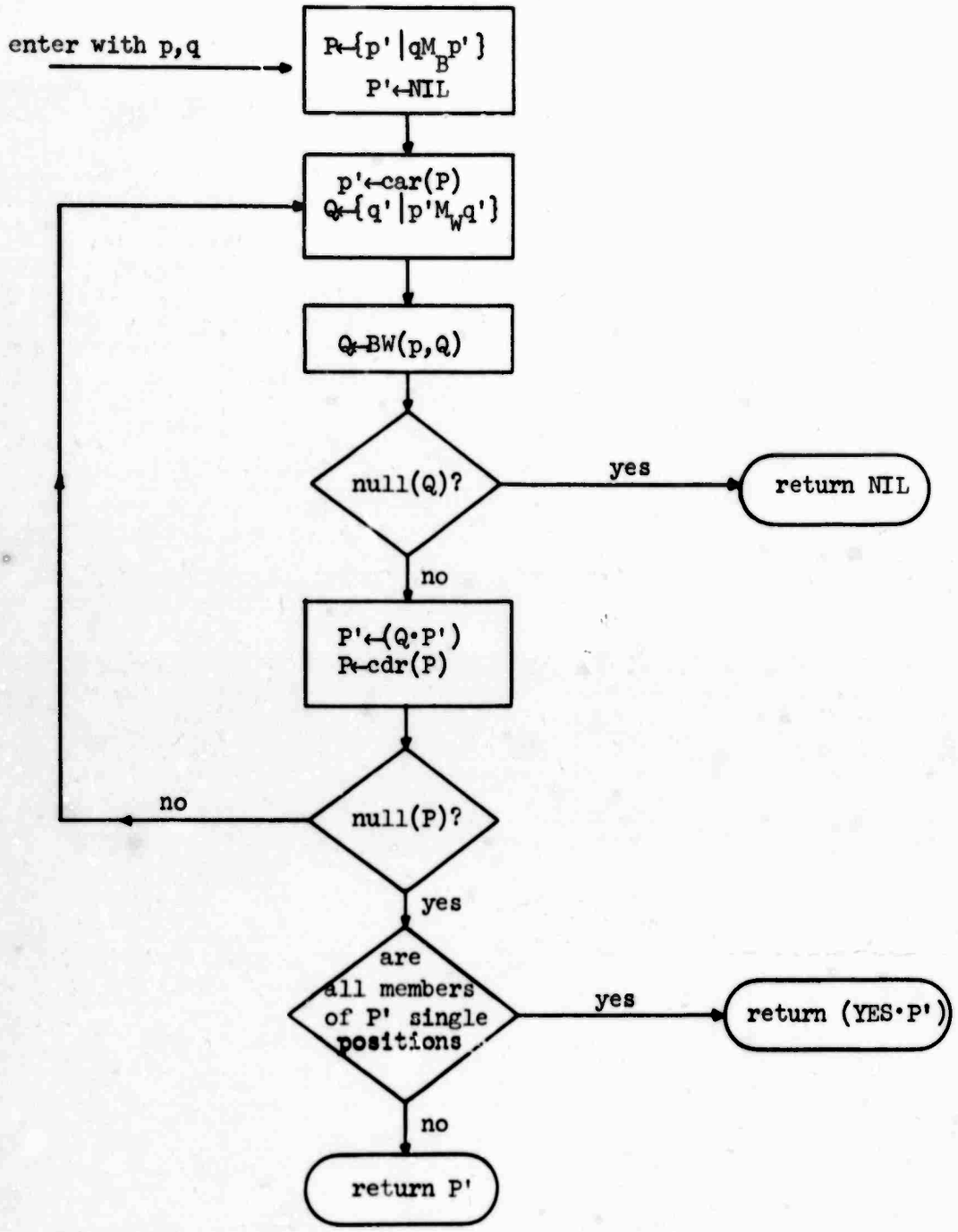


Figure 2.5. Function MB(p,q).q is a single position. Three values are returned.

(1) NIL means that some black move from q cannot be answered.
(2) YES•P' means that a <u>better</u> position is found for each black move from q.
(3) P' means that for at least one black move no decision has been made.

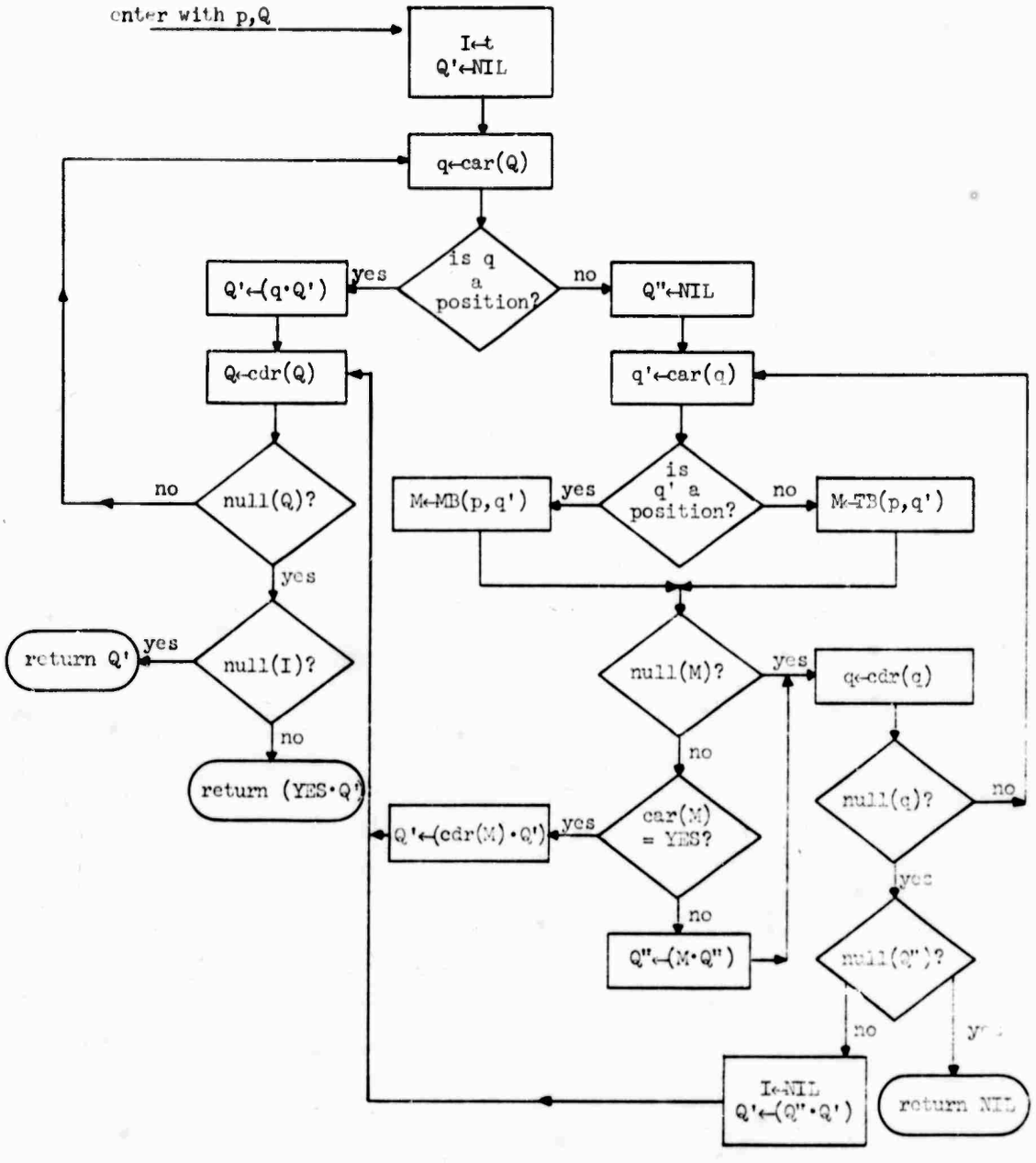


Figure 2.6. Function TB(p,Q). Q is a list consisting of positions and of lists of positions. If an element of Q is a single position, then it was found to be <u>better</u> at the previous level. If the element is a list of positions, these are the non-<u>worse</u> positions from the previous level. Q contains an element for each black move in the position immediately above in the tree.

Function TB returns

NIL - each member of a list of positions which is a element of Q is rejected in the search.

YES·Q' - all elements of Q are or lead to <u>better</u> positions.

Q' - some elements of Q do not lead to <u>better</u> positions.

Q' contains the tree from Q on down.

Tree Search Heuristics

Two heuristics are used during tree search. One helps to cut off redundant branches of the tree; the other helps the program find the better position faster.

1. Redundant Branch Cut-Off

Suppose we are down at a node at depth n in the tree. A history of the branch to this point is given by all the positions with black to move which the program has examined on the way to this node. There are n positions in this history, say q_i, q_{ij},..., $q_{ijk...m}$. At this point, suppose it is time to expand the node at the end of the branch. For simplicity let $q^* = q_{ijk...m}$. Now suppose that $BW(p, Q_1^*)$ returns a list of positions Q^*. The program checks the positions of the white pieces in each $q_i^* \in Q^*$ against the positions of the white pieces in q_i, q_{ij},..., q^*, and if there is a match, q_i^* is eliminated.

The reasoning behind this heuristic is as follows. It is true that two positions in which the white pieces are in the same squares but the black king is in a different square may have very different patterns. However, in this case one position is a successor of the other, and intuitively, if the placement of the white pieces is good, we should have taken advantage of this originally and done something else from there.

As far as the program is concerned, this heuristic has never caused it to miss a move it should have made. Part of the reason for this is that the trees are quite short (no more than a depth of seven) and within that short a span the intuition is probably valid. At

least one quarter of the positions returned by BW are eliminated by this heuristic.

2. Killer Heuristic

If in the tree a position $q_{ijk...m}$ is found to be <u>better</u> and not <u>worse</u> than p, the program finds out what the last white move, w, to $q_{ijk...m}$ was, and it remembers this move. Then every time after this, when it forms a set Q* to be used as an argument to BW, it checks to see if w was the last move made to form some $q^* \in Q^*$. If it was, then q* is made the first position in Q*, so that it will be examined first.

The theory is that in a tree search the positions are all similar, so a move which led to a <u>better</u> position at one point is likely to do so again. By putting the new position q* first we eliminate many comparisons if the theory holds. If the theory fails we have lost a little time.

In these end games the theory holds very well. If an examination is made of the final moves to the <u>better</u> positions during a tree search, usually there are only one or two such moves. The time saved when the position put first is actually the one selected is large enough to more than compensate for the time spent in ordering the positions.

<u>Representation</u>

No attempt has been made to develop a sophisticated representation for these end games. A position is represented by a list of the positions of the pieces. Moves are generated rather than stored. Patterns are

discovered by functions. Some information is very time consuming to obtain in this way, for example the set of all squares which a piece can reach in two moves. In general patterns of this type are not used, and the heuristics chosen for the end games reflect this.

CHAPTER 3

DEFINITION OF BETTER AND WORSE

As was explained in Chapter 1, each end game is played by different methods which we expect to result in different definitions of *better* and *worse*. However the form of *better* and *worse* is independent of the particular end games. In this chapter we will define the form, which will enable us to put a condition on the pattern recognition functions which make up *better* and *worse*. We will use this condition to prove that the program can reach checkmate from any starting position $p \in P$.

First of all, in order for the program to work correctly it must have a sense of direction. In the chess books this is achieved by an ordering of methods. For example in the rook end game, first we drive the opponent's king to an edge and then we keep our king on the same file (rank) as his. In the program, rules are represented by patterns of positions. Therefore the ordering of the heuristics is converted into an ordering of patterns, and positions from the end game can be grouped into subsets according to this ordering. Then a position q will be *better* than position p if the subset containing q is higher in the order than the subset containing p.

Recall that the program builds a forcing tree from a position p and then follows a branch of the tree (which branch is determined by the opponent's moves) until a position q at the end of the branch is reached. This position q is *better* than p. Now the opponent makes

The ending Rook and King against King.

The principle is to drive the opposing King to the last line on any side of the board.

In this position the power of the Rook is demonstrated by the first move, R–R7, which immediately confines the Black King to the last rank, and the mate is quickly accomplished by: 1 R–R7, K–Kt1; 2 K–Kt2.

The combined action of King and Rook is needed to arrive at a position in which mate can be forced. The general principle for a beginner

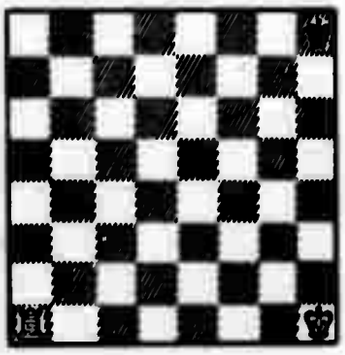


DIAGRAM 20

to follow is to keep his King as much as possible on the same rank, or, as in this case, file, as the opposing King.

When, in this case, the King has been brought to the sixth rank, it is better to place it, not on the same file, but on the one next to it towards the center.

2...K–B1; 3 K–B3, K–K1; 4 K–K4, K–Q1; 5 K–Q5, K–B1; 6 K–Q6.

Not K–B6, because then the Black King will go back to Q1 and it will take much longer to mate. If now the King moves back to Q1, R–R8 mates at once.

6...K–Kt1; 7 R–QB7, K–R1; 8 K–B6, K–Kt1; 9 K–Kt6, K–R1; 10 R–B8 mate.

It has taken exactly ten moves to mate from the original position. On move 5 Black could have played K–K1, and, according to principle, White would have continued 6 K–Q6, K–B1 (the Black King will ultimately be forced to move in front of the White King and be mated by R–R8); 7 K–K6, K–Kt1; 8 K–B6, K–R1; 9 K–Kt6, K–Kt1; 10 R–R8 mate.

Figure 3.1. Example from Capablanca, pages 26-28.

a move, giving position p'. At this point the program will build a forcing tree from p'. It does this without memory of positions p and q. If the program is to work correctly, it must be able to derive information about the state of the game from p', and any q' at the end of a branch of the forcing tree from p' must be better than p in addition to being better than p'. If this is true then we say the program is playing consistently. Consistency is accomplished by being careful about the selection of q in the first tree; however we must remember that only a moderate amount of tree search to q is permitted.

In the following section we will have much more to say about better than worse. This is not surprising, since for the program to work correctly worse need only recognize disaster and not interfere with better. Both of these conditions will be satisfied.

Formalization

The notion of a stage has been adopted to facilitate the program's sense of direction. The positions in an end game are divided into a number of different subsets called stages. The stages are not necessarily disjoint; however all the positions in a stage share a common pattern. In general a stage contains both positions with white to move and positions with black to move. The stages must exhaust the universe of positions in the end game. Let

$$Q = P \cup \{q \mid \exists p(p \in P \wedge pM_w q)\},$$

for P the set of all legal positions from which white can win. Every position $x \in Q$ must be in at least one stage. The stages are ordered, from the lowest (zero) stage containing stalemate positions and other

positions from which white cannot win, to the highest stage containing checkmate positions. The nth stage in the order is called stage n .

For programming purposes we prefer to deal with disjoint subsets. If $x \in Q$, we define

$\underline{st}(x) = 0$ if $x \in$ stage 0 .

$= \max(\{n \mid x \in \text{ stage } n\})$ if $x \notin$ stage 0 .

The subsets $\{x \mid \underline{st}(x) = n\}$ can be ordered by the value of $\underline{st}$ when applied to the elements of the subsets. These subsets are used to give the program a sense of direction in a natural way by

3.1 $\underline{st}(q) > \underline{st}(p) \supset \underline{better}(p,q)$.

Also we will have

$\underline{better}(p,q) \supset \underline{st}(q) \geq \underline{st}(p)$.

The statement

3.2 $\underline{st}(q) > \underline{st}(p) \equiv \underline{better}(p,q)$

is not used because it would result in tree searches of immoderate length.

3.1 is a partial definition of better, so we consider what condition is required to ensure that the program works consistently. Recall that we want to be able to deduce from the successors of q information about the state of the game at q . Suppose for now that 3.2 is the definition of better. Then the program can be forced to play consistently by the condition on stage definitions.

3.3 $\forall p' \, \forall q[qM_B p' \supset \underline{st}(p') \geq \underline{st}(q)]$.

3.3 says the stages must be defined in such a way that black can never force a return to a lower stage. This embodies the spirit of these

games; that is, that white is in complete control, and that the black moves are considered (by the program/student) only as part of the white strategy. We need not worry about a black move strategy.

There is no condition similar to 3.3 for white moves. However

3.4 $\forall p\ \exists q(pM_w q \wedge \underline{st}(q) \geq \underline{st}(p))$

is often useful. Intuitively it would seem that if some p had all successors at a lower stage, then p was evaluated incorrectly. This is not always true, but if 3.4 is not satisfied it is important to understand why.

As far as worse is concerned, we always have

$\underline{st}(q)=0 \supset \underline{worse}(p,q)$

which accomplishes branch termination and insures that worse recognizes disaster. We do not have

$\underline{st}(q)<\underline{st}(p) \supset \underline{worse}(p,q)$

because sometimes the path that the program should follow involves this kind of situation. We will always have

$\underline{worse}(p,q) \supset \underline{st}(q) \leq \underline{st}(p)$,

since worse may not interfere with better.

To help explain the definitions given in this chapter, an example will be developed as we proceed. It covers the play of part of the Rook and King against King end game, as explained in Capablanca [1935]; the text is given in Figure 3.1. This example can be handled in five stages. First we introduce pattern recognition functions f and g. For x a position, we have

$f(x) \equiv$ {the black king is confined to a file (rank) edge in x} .

Let $\underline{edge}(x)$ be the edge to which the black king is confined in x.

$g(x) \equiv \{f(x) \wedge$ (the white king is on the file (rank) two away from the file (rank) edge containing the black king and on a rank (file) closer to the center of the board than the black king)$\}$.

$f(x)$ represents the first principle in Figure 3.1. $g(x)$ partly represents the second principle in Figure 3.1; it will be used to recognize white move 6.

Now we can define the stages. These definitions are built up out of basic functions and notation which are described in Appendix A.

$x \in$ stage $0 \equiv \{x$ is stalemate, or x is a position with black to move, and black can take a white piece in one move$\}$.

$x \in$ stage $1 \equiv \{x$ cannot be assigned to any other stage$\}$.

$x \in$ stage $2 \equiv \{f(x) \wedge \underline{de}(wk_x, \underline{edge}(x)) > 2\}$.

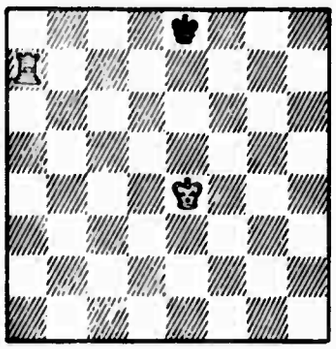
$x \in$ stage $3 \equiv g(x)$.

$x \in$ stage $4 \equiv \{x$ is checkmate$\}$.

Figure 3.2 gives examples of some of these stages. The opening position in Figure 3.1 is in stage 1. Note that every legal position (every position in set Q) is in some stage, because of the definition of stage 1. In every end game there will be a catch-all stage defined like stage 1.

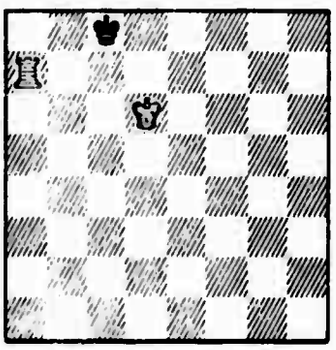
Now we must check that $\underline{st}$ satisfies 3.3. If $\underline{st}(q) = 2$ or $\underline{st}(q) = 3$, then the black king can never move in such a way as to form a p with $\underline{st}(p)<2$. This is because in q the black king is confined to an edge, and the white king is not blocking the rook since it is two or more files (ranks) away from the edge while the rook is

x_1



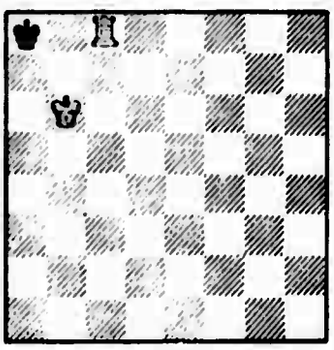
$\underline{st}(x_1) = 2$. This is the position after white move 4.

x_2



$\underline{st}(x_2) = 3$. This is the position after white move 6. Note that x_2 is in both stage 2 and stage 3.

q_3



$\underline{st}(q_3) = 4$. This is the checkmate position.

Figure 3.2. Stages in Figure 3.1.

only one away. The black king is not threatening to take the rook in any q with $\underline{st}(q)>0$, because in that case we would have $\underline{st}(q)=0$.

Rule 3.4 is also easy to satisfy. In stage 1 there is no difficulty. In stages 2 and 3, the rook will always be able to move to another square on the same file (rank) (for a file (rank) edge) and thus preserve the same stage.

If we use 3.2 as our definition of *better* and define *worse* by

$$\underline{worse}(p,q) \equiv \underline{st}(q)=0 ,$$

then only moves 1, 6 and 10 from the example in Figure 3.1 will be chosen by *better*. Thus the tree searches are fairly long, and also the tree is very wide. This brings us to the remainder of the definitions of *better* and *worse*. If we change the definition of *better* to

$$3.5\quad \underline{better}(p,q) \equiv \{\underline{st}(q)>\underline{st}(p) \vee [\underline{st}(q)=\underline{st}(p)=2 \wedge \underline{de}(wk_q,\underline{edge}(q))<\underline{de}(wk_p,\underline{edge}(p))]\}$$

then moves 1, 2, 3, 4, 5, 6, 10 will be recognized by *better*. This is a considerable improvement in the length of the tree search.

What is happening here in stage 2 happens in the other end games as well. The stage itself is rather large, but the positions inside it can be put into subsets, just as the whole universe of positions Q was put into stages. In fact, additional stages could be added, one for each of these new subsets.

However, we must consider an interesting property of the stages as they are defined in this end game, and one that is worth preserving in other end games. Recall that each stage is defined by a distinct pattern; in addition each stage is associated with its own heuristics.

Each stage has as its immediate goal the achievement of the next stage and its heuristics are directed toward that end. For example, in stage 2 we move the white king up toward the edge until stage 3 is reached; in stage 3 we force the black king toward a corner until checkmate is given.

If new stages were added for all these subsets, this heuristic property would be lost. While we may expect to use additional heuristics for two positions in the same subset of a stage, these heuristics are independent of the particular subset and can be used for all subsets within that stage. So it makes sense to handle these subsets differently from the stages. Therefore a new function has been added which is called a measure. For each stage n, function m_n is defined for all positions in stage n. m_n is not meaningful for every stage; in that case we have

$$m_n(x) = 0 \qquad \forall x(x \in \text{stage } n) .$$

Definition 3.5 implies the following measures

$$m_2(x) = \underline{de}(wk_x, \underline{edge}_x) \qquad \forall x(x \in \text{stage } 2) .$$

$$m_i(x) = 0 \qquad \forall x(x \in \text{stage } i) , \; i = 0, 1, 3, 4 .$$

Note that the smaller the measure, the better the position. This is the opposite of stages. Then the new (and complete) definition of *better* is

3.6 $$\underline{better}(p,q) \equiv \{\underline{st}(q) \rhd \underline{st}(p) \vee [\underline{st}(q) = \underline{st}(p) \wedge m_{\underline{st}(q)}(q) < m_{\underline{st}(q)}(p)]\} .$$

For program consistency, 3.3 becomes

3.7 $$\forall p \, \forall q\{q \, M_B p \supset [\underline{st}(p) \rhd \underline{st}(q) \vee (\underline{st}(p) = \underline{st}(q) \wedge m_{\underline{st}(p)}(p) \leq m_{\underline{st}(q)}(q))]\} .$$

An addition is also made to give the complete definition of worse. We have

3.8 $\underline{\text{worse}}(p,q) \equiv \{\underline{st}(q)=0 \vee [\underline{st}(p)=\underline{st}(q) \wedge m_{\underline{st}(p)}(p)<m_{\underline{st}(p)}(q)]\}$.

We can use this strong definition because if we have two positions in the same stage we know better how to compare them than if they come from different stages. We extend 3.4 to

3.9 $\forall p\ \exists q\ \{p\ M_w q \wedge (\underline{st}(q)>\underline{st}(p) \vee [\underline{st}(q)=\underline{st}(p) \wedge m_{\underline{st}(q)}(q)\leq m_{\underline{st}(p)}(p)])\}$.

Like 3.4, 3.9 is not necessary to the consistency of the program.

So far in this example stages have been defined in the same way for positions with white and black to move, excepting stage 0 and stage 4 which only contain positions with black to move. In general, however, slightly different versions of the same pattern are used to recognize positions with white to move as part of a stage than are used for positions with black to move.

For example, 3.6 selects white moves 1, 2, 3, 4, 5, 6, and 10 in Figure 3.1, but these are not the only moves it would select. In general we are not too concerned if the program doesn't select the book move, because the program is looking for a better position and not a best move. However in this case the program is playing differently from the book; it doesn't follow the second principle in Figure 3.1 and white moves 2 through 5 are affected by this. If we define

$x \in$ stage 2 $\equiv \{f'(x) \wedge \underline{de}(wk_x, \underline{edge}(x))>2\}$,

where

$f'(x) \equiv \{f(x) \wedge$ (the two kings are on the same rank (file) in $x)\}$,

then we will violate 3.7. For instance after move 1 in Figure 3.1, we have $f'(q)$; then the black king makes its move and we have $\neg f'(p)$.

What is needed is to define stage 2 differently for positions with white and black to move. We will use

$$x \in \text{stage } 2 \equiv \{f''(x) \wedge \underline{de}(wk_x, \underline{edge}(x)) > 2\} ,$$

where

$f''(q) \equiv f'(q)$

$f''(p) \equiv \{f(p) \wedge$ (the kings are on the same rank (file) or on adjacent ranks (files) in p)$\}$.

With this new definition of stage 2 the program will chose moves 2, 3, 4 and 5 correctly independent of the order in which the moves are generated. Another effect of the new definition is to put more positions in stage 1. In reality stage 1 would be divided into two or more stages, but here we are concerned only with the part of the end game covered in Figure 3.1.

Additions to better and worse

When functions are actually written for the play of end games, 3.6 will be the form for *better* and 3.8 for *worse*. However, certain additions will have to be made to *better* and *worse* to make the program practical. These additions will be made in the following format.

If the tree search is too long, then an addition to *better* is required. This will always have the form (for fixed n)

3.10 $(\underline{st}(p)=\underline{st}(q)=n \wedge \ldots)$.

We assume $m_n(p)=m_n(q)$ since $m_n(p)<m_n(q)$ would have been worse, and $m_n(p)>m_n(q)$ would already have been better. If the tree search is too broad, an addition will be made to worse. This will always have the form (for fixed n)

3.11 $\{\underline{st}(p)=n \wedge [\underline{st}(q)<n \vee (\underline{st}(q)=n \wedge m_n(q)=m_n(p))] \wedge \ldots\}$.

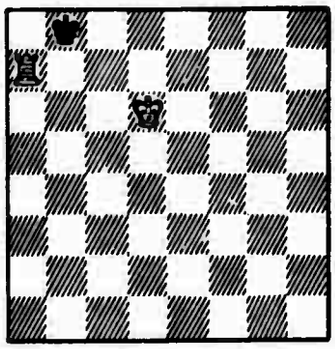
To be sure that the program will work consistently it is always necessary to extend 3.8 to cover additions 3.10, and 3.11 must not eliminate all former paths to better positions. Program consistency must be considered separately for each addition.

As an example of additions consider 3.6 and 3.8 as they apply to Figure 3.1. The definition of better is sufficient for this end game, so no problem of consistency arises. However the definition of worse needs to be enlarged. After move 6 in Figure 3.1, a tree of depth 4 is required to reach checkmate. Position p_1 in Figure 3.3 appears at the head of this tree. At the first level alone, 12 white moves are considered, and similar large numbers at further levels. If worse is changed to

$$\begin{aligned}\underline{worse}(p,q) \equiv \{&\underline{st}(q)=0 \vee [\underline{st}(q)=\underline{st}(p) \wedge m_{\underline{st}(p)}(q)>m_{\underline{st}(p)}(p)]\\ &\vee [\underline{st}(p)=3 \wedge \underline{st}(q)\leq 3 \wedge (d_q(wk,r)>d_p(wk,r)\\ &\vee [\underline{st}(q)\neq 3 \wedge d_q(wk,r)>1])]\}\end{aligned}$$

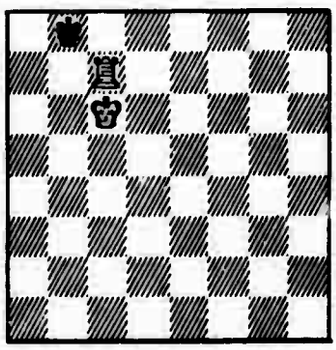
then only 4 moves are considered in p_1 . In p_2 , five out of nine moves remain; note that the desired move, wk-QKt6 gives q_2 not in stage 3 (see Figure 3.3). This tree is still rather broad and other or different heuristics can be added to prune more.

p_1



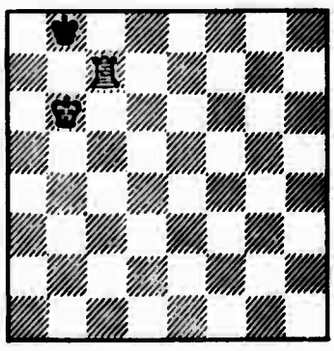
This is the position after black move 6 in Figure 3.1.

p_2



This is the position after black move 8 in Figure 3.1.

q_2



After white move 9; note that $\underline{st}(q_2) = 1$.

Figure 3.3

Formal Definitions of better and worse

3.6 $\underline{\text{better}}(p,q) \equiv \{\underline{st}(q)>\underline{st}(p) \vee [\underline{st}(q)=\underline{st}(p) \wedge m_{\underline{st}(q)}(q)<m_{\underline{st}(q)}(p)]\}$.

3.8 $\underline{\text{worse}}(p,q) \equiv \{\underline{st}(q)=0 \vee [\underline{st}(p)=\underline{st}(q) \wedge m_{\underline{st}(p)}(p)<m_{\underline{st}(p)}(q)]\}$.

Conditions on Stages and Measures

3.7 $\forall p \forall q\{q\ M_B p \supset [\underline{st}(p)>\underline{st}(q) \vee (\underline{st}(p)=\underline{st}(q) \wedge m_{\underline{st}(p)}(p)\leq m_{\underline{st}(q)}(q))]\}$.

3.9 $\forall p \exists q \{p\ M_W q \wedge (\underline{st}(q)>\underline{st}(p) \vee [\underline{st}(q)=\underline{st}(p) \wedge m_{\underline{st}(q)}(q)\leq m_{\underline{st}(p)}(p)])\}$.

Additions to the Formal Definitions of better and worse

Additions to better have the form

3.10 $(\underline{st}(p)=\underline{st}(q)=n \wedge \ldots)$.

Additions to worse have the form

3.11 $(\underline{st}(p)=n \wedge [\underline{st}(q)<n \vee (\underline{st}(q)=n \wedge m_n(q)=m_n(p))] \wedge \ldots)$.

Figure 3.4 Listing of the Rules Introduced in Chapter 3.

CHAPTER 4

ROOK AND KING AGAINST KING

Formal Definitions of better and worse

The method of play chosen for this end game is taken primarily from Fine [1942]. His description is given in Figure 4.1. The last few moves of the game are chosen by Capablanca's [1935] method illustrated by moves 8-10 in Figure 3.1.

Only one basic pattern, shown in position q_1 in Figure 4.2, is required for this method of play. The ability of the rook to control ranks and files is utilized; as long as the black king is not in check it is held in some area of the board by the rook. Usually this area is a quadrant as shown in q_1. If the white king is not on the boundary of the area, the black king can escape only by attacking the rook. If the white king is outside of the area, as shown in q_1, it is able to protect the rook from such an attack if it is close enough. It can't be blocked from protecting the rook by the black king.

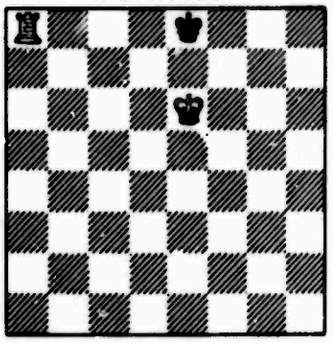
If the pattern shown in q_1 holds in a position, this is recognized by function quad:

quad(x) ≡ (the rook confines the black king to an area of the board in x, and the white king is outside that area).

quad describes the pattern occurring in almost all positions of Figure 4.1. For example quad holds after each of the first three black and white moves. If quad is satisfied by a position, we will refer to

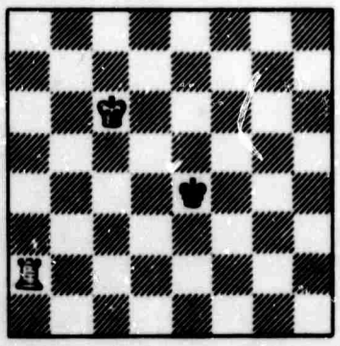
This piece is not nearly as strong as the Queen and the mate is accordingly far more difficult. The Rook alone cannot drive the King to the edge of the board—it needs the assistance of its own monarch. Since the Rook is much less powerful than the Queen, there is less danger of stalemate—this is the brighter side of the picture.

Mating Position with the Rook.

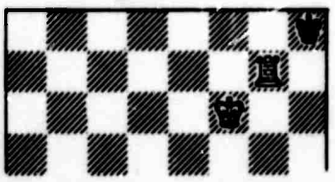


In order to mate, the enemy King must again be driven to the edge of the board. The mating position is then the same as the second one with the Queen. Thus the problem here is essentially the same as that in the previous case, the chief difference being that the two preliminary steps (driving the enemy King back and bringing one's own King up) are carried out simultaneously. The only stalemate that should be watched for occurs

No. 2



Black to Play is Stalemated.



when the Black King is in the corner.

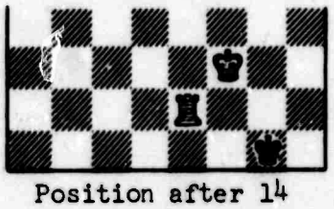
Starting from any position such as that shown here in No. 2 we would then proceed as follows: 1R-Q2 (confining the Black King to the right-hand side of the board), 1 .... K-K6; 2 R-Q5, K-K5; 3 K-B5, K-K6; 4 R-Q4 (now he has only three ranks and four files), K-K7; 5 K-Q5, K-K6; 6 K-K5, K-K7; 7 K-K4, K-B7; 8 R-Q3 (see diagram No. 2a), K-K7; 9 K-Q4, K-B7; 10 R-K3, K-Kt7; 11 K-K4, K-B7; 12 K-B4, K-Kt7; 13 R-K2ch, K-B8; 14 K-B3, K-Kt8 (diagram No. 2b); 15 K-Kt3, K-B8; 16 R-K8, K-Kt8; 17 R-K1 mate.

The final maneuver, which involves losing a tempo, or move, should be remembered—it is the key to this mate.

No. 2a

Position after 8 R-Q3.

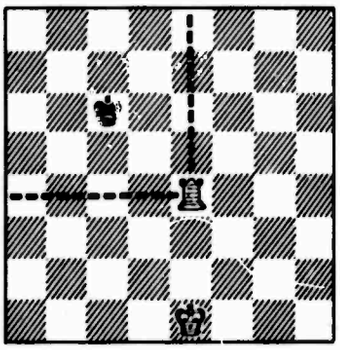
No. 2b



Position after 14 .... K-Kt8.

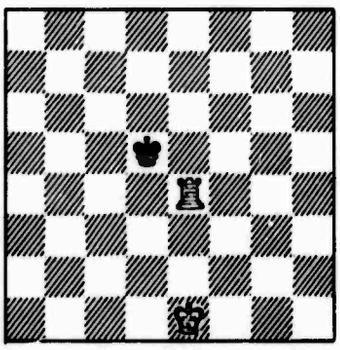
Figure 4.1. Example from Fine, pages 14 and 15.

q_1



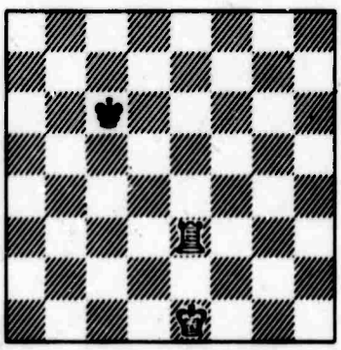
We have quad(q_1) and squad(q_1) = 16.

p_2



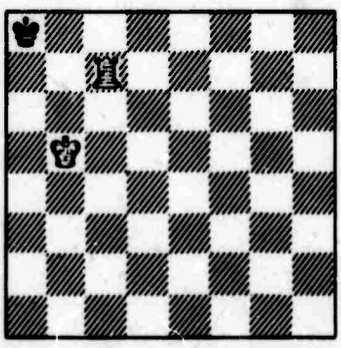
We still have quad(p_2) but we must move the rook or we will lose it. The rook can move so that there will be a quadrant, but the size will be larger.

p_1



Here we have quad(p_1) and squad(p_1) = 20. We have $p_1 M_W q_1$. We do not want to accept q_1 as better than p_1.

x_2



Here we have quad(x_2) and squad(x_2) = 2. x_2 is in stage 3.

Figure 4.2

the area in question as a quadrant. This pattern lends itself very naturally to a measure. If we have quad(x), then squad(x) is the number of squares inside the quadrant. For example, in q_1 in Figure 4.2, squad(q_1) = 16.

If quad is to be used to determine a stage and squad is to be a measure in that stage, we must satisfy conditions 3.7 and 3.9 (see Figure 3.4). Condition 3.7 presents no problem since both quad and squad depend only on the positions of the white pieces. The black king is unable to escape from a quadrant except by taking the rook; in this case the position q prior to the black move would be in stage 0.

Rule 3.9 cannot be satisfied without putting additional conditions in the stage definition. For example, suppose in position q_1 the black king moved to Q4 to attack the rook, forming p_2 in Figure 4.2. The white king is not close enough to protect the rook; therefore we must move the rook away from the black king. It is simple to form a new quadrant; for example, any rook move on the fourth rank will do this. However every rook move which forms a quadrant forms one of a larger size. In general, the rook can always form a quadrant, but it may be larger than the present one. This violates rule 3.9.

Note that it really would be incorrect for the program to accept a position like q_1 as <u>better</u> than, for example, p_1 in Figure 4.2. At position p_2, the best that white can do is to maintain the smallest possible quadrant. This will have size 20, the same as squad(p_1). Therefore nothing has been gained by making the move to q_1 and the burden of correct play has been pushed onto the tree search.

Now the problem in position q_1 came about only because the white king was too far away from the rook to protect it from the black king's attack. Therefore all that is needed to satisfy rule 3.9 is to insist that the white king protect the rook. The condition of protection is given by function goodquad

$$\underline{\text{goodquad}}(p) \equiv \{\underline{\text{quad}}(p) \wedge d_p(wk,r) \leq d_p(bk,r)+1\}$$

$$\underline{\text{goodquad}}(q) \equiv \{\underline{\text{quad}}(q) \wedge d_p(wk,r) \leq d_q(bk,r)\} .$$

Different definitions are given for p and q to insure that goodquad satisfies 3.7. (We remind the reader that definitions of basic functions and notation are given in Appendix A.)

The use of goodquad for a stage and squad for a measure in that stage will inexorably force the black king toward a corner of the board. However, this process must stop when we reach a quadrant of size 2, since any smaller quadrant would be stalemate. Therefore when squad = 2 we must move to a new stage. At this point we shift to the heuristics taken from Capablanca [1935]. x_2 in Figure 4.2 is an example of a position from this stage (stage 3).

We give the formal definition of better and worse by defining the stages and measures.

x ∈ stage 0 ≡ (x is stalemate or x is a position with black to move and black can take the rook in one move).

x ∈ stage 1 ≡ x cannot be assigned to any other stage.

$$x \in \text{stage } 2 \equiv \{\underline{\text{goodquad}}(x) \wedge \underline{\text{squad}}(x) > 2\} .$$

$$x \in \text{stage } 3 \equiv \{\underline{\text{goodquad}}(x) \wedge \underline{\text{squad}}(x) = 2\} .$$

x ∈ stage 4 ≡ x is checkmate.

Only stage 2 has a meaningful measure. We have

$m_2(x) = \underline{squad}(x)$ $\quad \forall x(x \in \text{stage } 2)$

$m_i(x) = 0$ $\quad i \neq 2$.

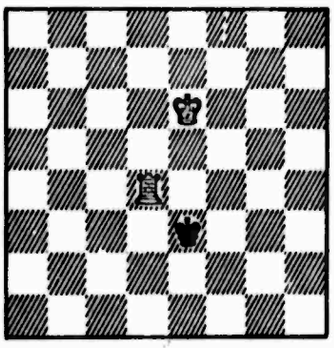
Additions to better and worse

We are now ready to consider how well the program plays using the formal definition of better and worse. We need not worry about the transition from stage 1 to stage 2, since the tree search is no greater than depth 2. However, the depth of tree search in stage 2 can be as large as 8, although a depth of 3 is average; in stage 3 there is a maximum depth of 5. Therefore, we must make additions to better and worse in stages 2 and 3.

In stage 2 both the length and the width of the tree must be reduced. Recall that we are striving to shrink the size of the quadrant. The rook alone is unable to do this; sometimes the white king must be used to force the black king away from the rook. For example, in p_2 in Figure 4.3 the white king must move onto the boundary of the quadrant. Then on the next white move the rook can form a new quadrant smaller than the present one (see position q_3 in Figure 4.3). In order for the white king to be useful, it must first be next to the rook. Position p_1 in Figure 4.3 is an example of a position in which the white king must move up to the rook. We can recognize this kind of move by adding to better

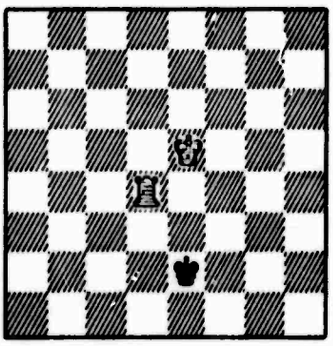
4.1 $\underline{st}(p)=\underline{st}(q)=2 \wedge d_q(wk,r)<d_p(wk,r)$.

p_1



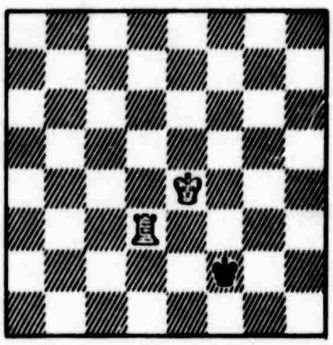
In this position obviously we want wk-Q5 or wk-K5.

p_2



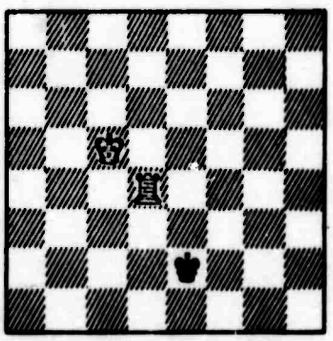
This is the position before move 6 in Fine (Figure 3.1). Now we want to move the white king onto the boundary to force the black king away from the rook. The move made in Fine is wk-K4; wk-KB4 is just as good.

q_3



The black king is forced to move away from the rook (bk-KB7 in Fine), and then the rook can form a smaller quadrant (r-Q3), giving q_3.

p*



p*, taken from Figure 4.1 before white move 5, is the start of the longest tree (depth 4).

Figure 4.3. Examples of Moves in Stage 2.

4.1 reduces the length of the tree search to a maximum of four. This is a manageable length so no further change need be made to *better*.

A tree search of depth three or four requires considerable pruning to be practical. In the formal definition only rook moves leading to larger quadrants and moves giving stage 0 are eliminated. In p^* in Figure 4.3, for example, seven white king moves and four rook moves would be examined in the tree search. This tree will be too broad.

Note first of all that tree search will take place only when $d_p(wk,r) = 1$. The strategy at this point is to move the white king onto the boundary, which gives a position q in stage 1. Therefore not all stage 1 positions q can be declared *worse* than p. However, the rook can also move to form a stage 1 position, either by moving so that in q there is no quadrant or the rook is not protected by the white king. All these moves can be eliminated. In addition all white king moves which result in $d_q(wk,r) > 1$ can be eliminated. We add to *worse*

4.2 $\underline{st}(p)=2 \wedge [\underline{st}(q)<2 \vee (\underline{st}(q)=2 \wedge m_2(p)=m_2(q))]$

$\wedge [d_p(wk,r)=1]$

$\wedge [d_q(wk,r)>1 \vee (\underline{st}(q)=1 \wedge r_p \neq r_q)]$

It is easy to see that these additions to *better* and *worse* are correct. First we note that

$$(qM_Bp \wedge \underline{st}(q)=2) \supset (\underline{st}(p)=2 \wedge m_{\underline{st}(p)}(p) = m_{\underline{st}(p)}(q) \wedge d_p(wk, r) = d_q(wk, r)) .$$

Therefore 3.7 can be extended to cover 4.1. As far as 3.9 is concerned, the important thing is that the white king is always able to move to

protect the rook and such a move will insure

$$d_q(wk,r) \leq d_p(wk,r) .$$

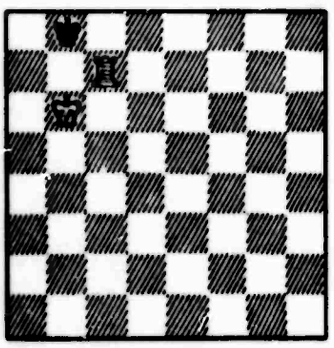
We note also that 4.1 can only be applied to finite number of times (no more than 7) between applications of the formal definition of better. 4.2 is correct because it does not interfere with 4.1 or the formal definition of better, even when a tree search is required to force a smaller quadrant.

In stage 3, the maximum length of the tree sea.ch is five, so it is not necessary to change better. However considerable tree pruning will be needed to make the tree manageable.

The checkmate position is illustrated by q_3 in Figure 3.2. Before the checkmate can be given, the white king must be in the square indicated in q_3. Note $d(wk,r) = 1$ in the checkmate position. Now we could have used $d_x(wk,r)$ as a measure in stage 3 but it leads to considerable inaccuracy of play since only the indicated square, of all the squares next to the rook square, is used for checkmate. We have concentrated instead on tree pruning.

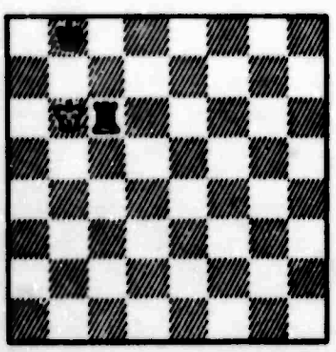
Although we do not use $d(wk,r)$ as a measure, it is obvious that we do not want to move the white king away from the rook. This one rule will eliminate many king moves. However, the rook also contributes many moves, some rook moves giving stage 2 positions and some stage 1. The stage 2 moves can be eliminated, but sometimes a stage 1 rook move is necessary. This case is illustrated by position p_1 in Figure 4.4. At this point the rook must make a "tempo" move. It must remain on the QB file, so that the black king is forced to move into the corner. However, there are six usable squares on that file. We can limit the

p_1



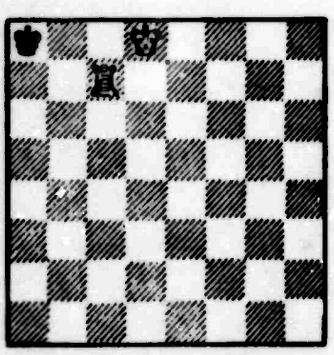
No immediate successor position is <u>better</u> than p_1. QB6 is a desired move

q_1



After r-QB6, we have q_1 in stage 1. bk-R1 is the only legal move, and then we can get checkmate. This rook move is needed for parity.

p*



p* is an example of the longest tree search, and the depth of this tree is 5 moves. It is quite narrow, however. After pruning, the remaining white moves are r-Q7, r-K7, r-QB8 and wk-Q7.

Figure 4.4. Examples of Stage 3.

rook moves examined by insisting that the rook stay next to the white king. In stage 3 we add to worse:

4.3 $\underline{st}(p)=3 \wedge [\underline{st}(q)=2 \vee (\underline{st}(q)=1 \wedge d_q(wk,r)>1)$
$\vee (\underline{st}(q)=3 \wedge d_q(wk,r)>1 \wedge d_q(wk,r)\geq d_p(wk,r))]$.

4.3 is correct because again we have been careful not to eliminate all paths to checkmate. Now the tree is narrow enough to manage. In p_1 for instance only four moves are left after pruning; in p* also four moves are left. Since very few moves are available to the black king the tree remains quite narrow.

Combining formulas 4.1, 4.2 and 4.3 with the formal definitions of better and worse we have:

$$\underline{better}(p,q) \equiv \{(\underline{st}(q)>\underline{st}(p))$$
$$\vee (\underline{st}(q)=\underline{st}(p) \wedge m_{\underline{st}(q)}(q)<m_{\underline{st}(q)}(p))$$
$$\vee (\underline{st}(p)=\underline{st}(q)=2 \wedge d_q(wk,r)<d_p(wk,r))\} .$$

$$\underline{worse}(p,q) \equiv \{\underline{st}(q)=0 \vee (\underline{st}(q)=\underline{st}(p) \wedge m_{\underline{st}(p)}(p)<m_{\underline{st}(q)}(q))$$
$$\vee (\underline{st}(p)=2 \wedge [\underline{st}(q)=1 \vee (\underline{st}(q)=2 \wedge m_2(p)=m_2(q))]$$
$$\wedge d_p(wk,r)=1$$
$$\wedge [d_q(wk,r)>1 \vee (\underline{st}(q)=1 \wedge r_p\neq r_q)])$$
$$\vee (\underline{st}(p)=3 \wedge [\underline{st}(q)=2 \vee (\underline{st}(q)=1 \wedge d_q(wk,r)>1)$$
$$\vee (\underline{st}(q)=3 \wedge d_q(wk,r)>1$$
$$\wedge d_q(wk,r)\geq d_p(wk,r))])\} .$$

These are the functions actually used in the program.

Examples of Program Play

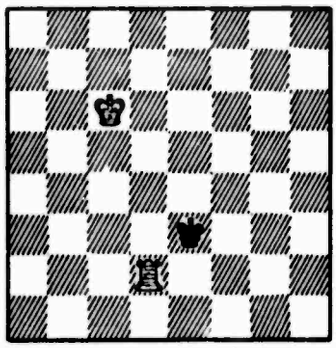
In order to prove that the program works we must give examples of program play. The first example is taken from Figure 4.1. The program

is started at the second move because it would make r -QR5 as its first move. The reason for this difference will be discussed later.

The opening position is p_1 in Figure 4.5. We have:

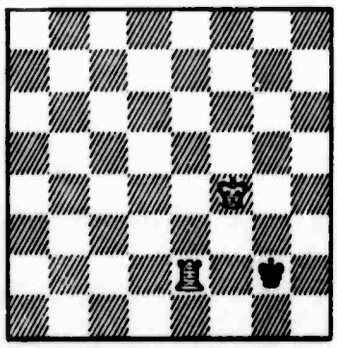
1.	r -Q8		p_1 is a stage 1 position, therefore the first stage 2 position generated is <u>better</u>.
		bk -K5	
2.	r -Q5	bk -K6	The program has lost one move.
3.	wk-Q6	bk -K5	
4.	wk-K6	bk -K6	
5.	wk-K5	bk -B6	
6.	r -Q4		Now we have <u>squad</u> = 16. The white moves 3 to 6 are chosen by a tree search.
		bk -K7	
7.	wk-K4	bk -B7	
8.	r -Q3	bk -K7	This is the same position as the book's after move 8. White moves 7 and 8 are chosen in a tree search.
9.	wk-Q4	bk -B7	
10.	r -K3	bk -Kt7	The tree has depth 3, but this branch (moves 9 and 10) is only depth 2.
11.	wk-K4	bk -B7	

p_1



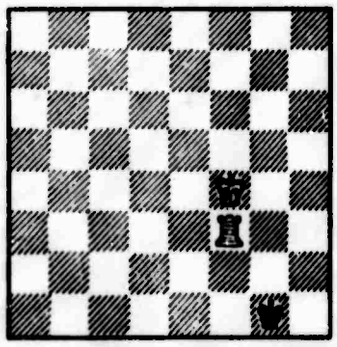
The opening position for the example is in stage 1.

q_2



The position after book move 13.

p_2



The rook and white king are in the position after move 13 by the program.

Figure 4.5. Illustrations of Examples of Program Play.

12.	wk-B4	bk -Kt7	
13.	r -B3		Moves 9-12 are the same as the book's, but now we differ.
		bk -R7	
14.	r -Kt3		We are now in stage 3; see Figure 3.1 at move 7.
		bk -R8	
15.	wk-B3	bk -R7	
16.	wk-B2	bk -R8	
17.	r -R3 mate		

The program is playing very similarly to the book up to move 13. It choses a <u>better</u> and not the best position at move 1, and then must work hard to catch up to the book. It is in a better position after move 6 than the book is after move 4 and is able to regain two moves. At move 13 the book makes a move using a different strategy. Instead of shrinking the quadrant it puts the king in check (see q_2 in Figure 4.5). If the black king goes to any square but B8 the book gives mate in two or three moves, but for the move to B8, four moves are required. The program's move also requires four more moves to checkmate, so it is really just as good as the book move.

Position p_2 in Figure 4.5 is the starting position for this next example. p_2 is the position which results if in the previous example we have

13.	...	bk -Kt8
14.	wk-Kt3	bk -R8

15. r -B2		r-B1 is checkmate, but r -B2 is generated first and also gives a better (stage 3) position.
	bk-Kt8	
16. r -B4	bk-R3	
17. r -B1		Checkmate.

However, the order of moves can also be correct. If

13. ...	bk-R8	
14. r -B2		and two moves to checkmate. r-Kt3 also gives a better position, but four moves would have been required to mate.

The numbering of the program moves is one less then it should be since the program started at book move two. This means the program never recovered the move it lost at its first move.

Evaluation of Program Play

Now we can see that the program plays similarly to the books. More important, it is using the same heuristics as the book's in most cases. For example, the use of squad as a measure exactly models Fine's book when it is concerned with cutting down the number of ranks and files available to the black king (see the comment after move 4 in Figure 4.1). Also both the program and the book use the white king to protect the rook and to force the black king away from the rook so a smaller quadrant can be formed.

The differences in program and book play that do occur illustrate features of program play. These will be discussed in detail in Chapter 7; only a list will be given here.

1. The goodness of program play is dependent on the order of move generation (illustrated by the last two game examples).
2. The program will accept a move which gives a <u>better</u> position at depth 1 even if an advantage would be gained by waiting until depth 2 to evaluate. This is the reason that the program will not make book move 1.
3. The program uses a single main heuristic inside a stage; it will not switch heuristics until it reaches a new stage. This is the reason the program will not make book move 13.

None of these features causes the program any serious difficulty. In fact, the program plays this end game very well. If it can do as well on other end games, we will be very satisfied with it.

CHAPTER 5

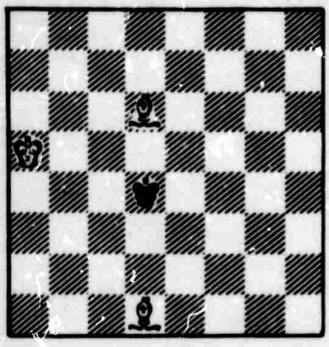
TWO BISHOPS AND KING AGAINST KING

This end game, while not difficult, is considerably harder to play than the Rook end game, and the increase in difficulty is mirrored in the program. The final definitions of <u>better</u> and <u>worse</u> are quite complicated. As in the Rook game, the method of play used is a composite of Fine [1944] and Capablanca [1935]. Figure 5.1 is the example from Fine; Capablanca's method is given in Figure 5.6. Again Fine's method is used to guide the first part of the game, while Capablanca's is used in the final stages.

Two basic patterns are sufficient for the entire end game. The first, as in the Rook end game, is concerned with confining the black king to an area of the board. Unlike the rook, a bishop does not hold an uncrossable line. However when two bishops are on adjacent diagonals they together do hold such a line. Position x_1 in Figure 5.2 illustrates this; the black king is confined to approximately half the board. When the bishops are also on adjacent squares, the space available to the black king is even smaller, approximately a quarter of the board. This is shown in positions x_2 and x_3 in Figure 5.2. In addition when the two bishops are on adjacent squares they may protect each other, as in x_2. If not, as in x_3, then only one bishop is open to attack, and there is only one square inside the area from which the black king can attack it. Therefore it is fairly easy for the

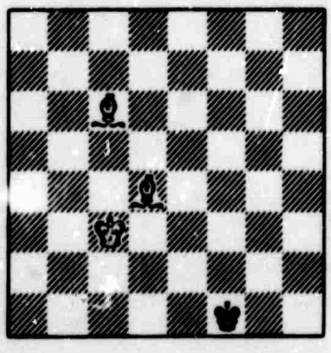
In the previous cases it has always been sufficient to drive the King to the edge of the board. Here, however, it is essential to have the enemy King in a corner, for though mating positions in the center are possible they cannot be forced. Any corner will do (unlike the case with Bishop and Knight).

No. 3



Beginning with any arbitrary position (see diagram No. 3) the first task is to reduce the mobility of the Black King. Thus 1 B-B3, K-K6; 2 B-B6, K-Q5. Now that the Bishops are as well placed as possible the King must come up. 3 K-Kt4, K-Q6; 4 B-K5, K-K6; 5 K-B4, K-Q7; 6 B-Q4, K-K7; 7 K-B3, K-B8 (see diagram No. 3a); 8 B-B3, K-K8; 9 B-Kt2, K-K7; 10 B-B5 (a tempo move: White cannot approach directly and loses a move to compel Black to retreat), K-K8; 11 K-Q3, K-Q8 (see diagram No. 3b). From this point on the rest is quite simple: by successively cutting off the squares to the right of Black he is compelled to play into the corner. 12 B-Kt4, K-B8; 13 B-KB3, K-Kt7; 14 B-Q1 (the King must not be allowed to escape), K-B8; 15 B-R4, K-Kt8; 16 K-Q2, K-Kt7; 17 K-Q1, K-Kt8; 18 B-B3, K-R7; 19 K-B2, K-R6; 20 B-Kt5, K-R7; 21 B-Kt4, K-R8; 22 B-Q3 (tempo move), K-R7; 23 B-B4ch, K-R8; 24 B-B3 mate.

No. 3a. Position after Black's 7th Move.



No. 3b. Position after Black's 11th Move.

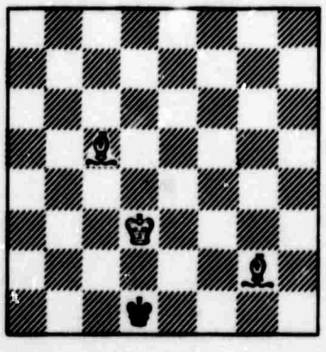
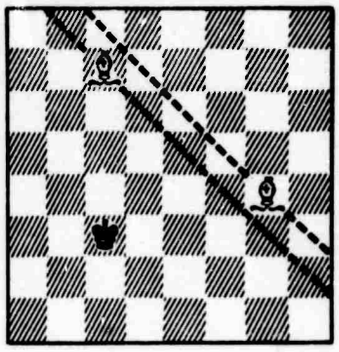


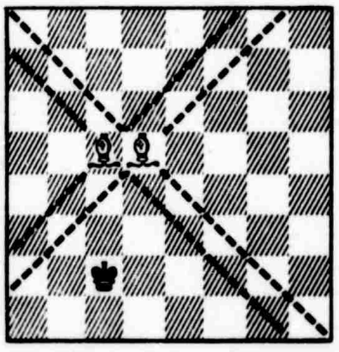
Figure 5.1. Example from Fine, p. 15-17. This method serves as a guide for the first part of the game.

x_1



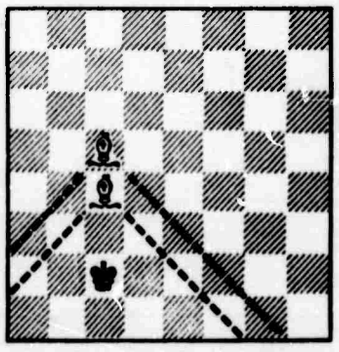
When the two bishops are on adjacent diagonals they confine the black king to approximately half the board.

x_2



If in addition they are on adjacent squares they confine the black king to approximately a quarter of the board. In x_2 they also protect each other.

x_3



However, the white bishop is open to attack in x_3.

Figure 5.2. Examples of Quadrants.

program to evaluate the danger of attack and decide how to prevent it. For this reason, together with the advantage of confining the black king to a small area, the program uses this configuration as its sole pattern. In Fine, this pattern is combined with the one where the two bishops are simply on adjacent diagonals. Capablanca does not describe the middle part of this end game; however the part he describes is a continuation of this method (see Figure 5.6).

Now when the bishops are side by side they keep the black king in approximately a quarter of the board, so this area will be called a quadrant. Such an area will be recognized by function quad(x) . In order for the black king to be confined to an area it must either be inside the area or else possibly on the inner diagonal of the boundary of the area. For example, in x_3 in Figure 5.2, squares QR2, QKt3, Q3, K2 and KB1 may be acceptable positions for the black king, in addition to the inside squares. We have

quad(x) ≡ {the black king is inside the area formed by the two adjacent bishops, or it is on the inner diagonal of the boundary of the area}.

Note that the position of the white king is not considered in quad.

It is easy to define a size for a quadrant. The area in which the black king is controlled by the bishops has the shape of a triangle, and an edge of the board forms the side of the triangle opposite the two bishops. Call this edge, edge(x) . Then

squad(x) = de(kb_x, edge(x))+de(qb_x, edge(x)) .

Thus squad(x_2) = 8 and squad(x_3) = 7 . (For definitions of basic functions and notation, refer to Appendix A.)

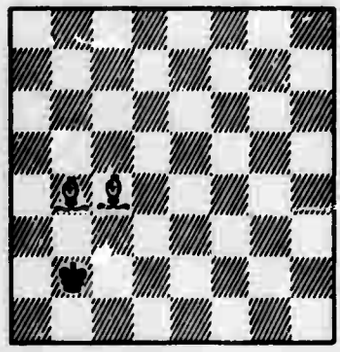
The fact that we intend to use Capablanca's method for the last part of the game puts a restriction on the quadrants the program uses in this stage (stage 2). Position x_1 in Figure 5.3 is an example of the start of Capablanca's method. Note that the quadrant in x_1 contains a corner. Now if we decrease the size of the quadrant indiscriminately we may end up with the black king confined to a small area not containing a corner, as in x_2 in Figure 5.3. Then we would have to use an intermediate heuristic to achieve x_1. Rather than do this we force x_1 to occur directly by only using quadrants containing corners. Function hascorner(x) is true if the quadrant in x contains a corner. This constraint makes it more difficult for the program to force a smaller quadrant, since often only one of the two immediately smaller quadrants contains a corner. Position p in Figure 5.3 is an example.

We now must consider whether quad and squad will satisfy conditions 3.7 and 3.9. For condition 3.7 we define

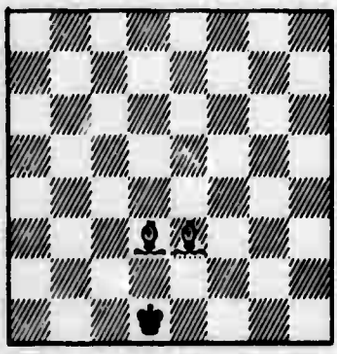
spec(x) ≡ {some successor of the black king in x is not inside the quadrant}.

Then for a quadrant to be accepted in q, spec(q) must be false. For example, q_1 in Figure 5.4 will be rightfully rejected by this condition; after the black king moves to KB7 no white move can force it back into the area. Requiring that spec(q) be false insures that the black king must move inside the quadrant, and any p with the black king inside the quadrant will be accepted. Therefore rule 3.7 is satisfied. Condition 3.9 presents more difficulty. First we must reject positions like p_2 in Figure 5.4. In p_2, only qb-Q6 will form a quadrant, but

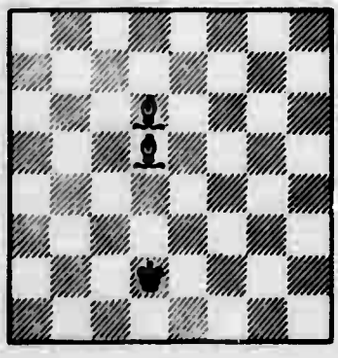
x_1



x_2



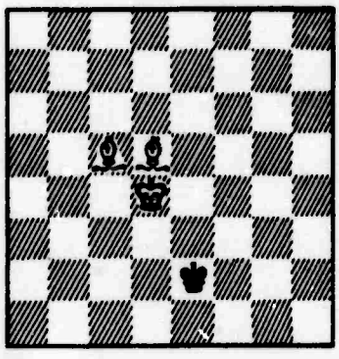
p



The quadrant in p has a corner. However, of the two ways of forming a smaller quadrant, only one, qb-QB5, produces a quadrant with a corner.

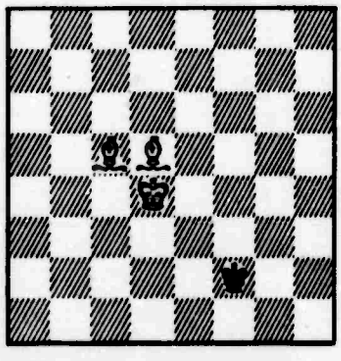
Figure 5.3.

q_1



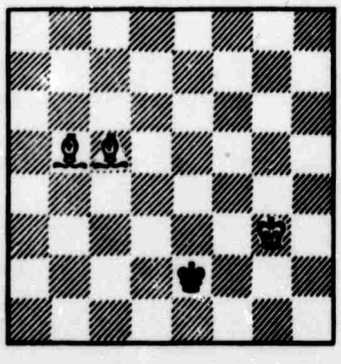
The black king can move to KB7, giving p_2,

p_2



and no white move can contain the black king in a quadrant of size 8.

q_3



The white bishop has just moved from QB6 (squad = 9). Now squad(q_3) = 8. The black king is controlled by the white king and must move inside the quadrant.

Figure 5.4.

this quadrant has size 9. If p_2 is considered to have a quadrant, then that quadrant would have size 8. Therefore, if in p the black king is on the boundary of the area, we insist that spec(p) be false if the quadrant is to be accepted.

We have now eliminated any possibility of direct black escape from the quadrant. However, we must consider whether white may be forced to give up an advantage because of a black threat. Now black can only threaten a bishop, and in a position like x_2 in Figure 5.2 the bishops protect each other. Therefore we need only worry about a quadrant like the one in x_3 in Figure 5.2. A position x with a pattern like x_3 can easily be recognized because squad(x) is odd. If this is true, the quadrant is called a head quadrant, and head(x) is the square containing the bishop closest to the black king. This bishop is referred to as the head bishop.

In the Rook end game we solved a similar problem by always insisting that the white king be close to the vulnerable piece. Here things are not so simple. In the Rook game the white king could assist in shrinking the quadrant from a square next to the rook, but in this end game the white king may have to move away from the head bishop in order to be of use. For example in q_3 in Figure 5.4 the king's bishop has just moved from QB6; prior to this move the queen's bishop on black square QB5 was the head bishop, and the white king is four away from this bishop.

If in a head quadrant we can make a move into a smaller non-head quadrant we have cancelled any threat the black king was making. If either bishop could move to make a smaller quadrant, then if

$d(bk_q, \underline{head}(q)) > 3$ we would always be sure of forming the non-head quadrant in time. However, because of the corner condition, usually only one bishop move is permitted. In this case the white king is the only sure means of defending the head bishop. However, if at any point we know we can form a smaller quadrant in time, we will take advantage of that fact.

It is difficult to be sure that the white king can protect the head bishop. In q_1 in Figure 5.5 we have $d(wk_{q_1}, \underline{head}(q_1)) = d(bk_{q_1}, \underline{head}(q_1))$, but even so the white king cannot protect the head bishop. Therefore in q we expect the condition

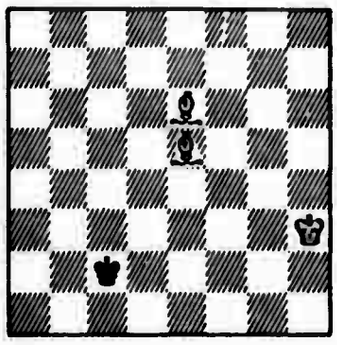
$$d(wk_q, \underline{head}(q)) < d(bk_q, \underline{head}(q))$$

and in p

5.1 $d(wk_p, \underline{head}(p)) \leq d(bk_p, \underline{head}(p))$.

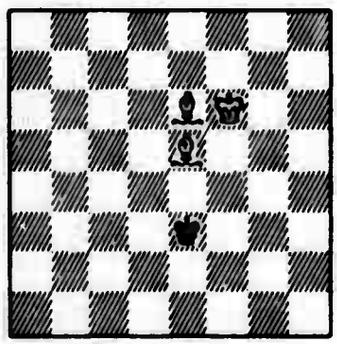
However these conditions are not even sufficient. Position p_3 in Figure 5.5 is an example. p_3 satisfies 5.1 but a bishop will have to move to form a larger quadrant because every white king move leaves the head bishop unprotected. This condition can be recognized in the position from which black moved to form p_3 (position q_2 in Figure 5.5). Note that the white king position shown in q_2 and p_3 is just one of many which are bad. The bad squares are: KKt3, KKt4, QB3, QB4, KB6, and Q6. Also all squares more than two away from the head bishop are bad. The remaining squares are good: they are KKt5, KKt6, KKt7, KB5, KB7, K7, Q5, Q7, QB5, QB6, and QB7.

One final case remains to be considered, and it is illustrated by position q_4 in Figure 5.5. We have $d(bk_{q_4}, \underline{head}(q_4)) = 2$ and

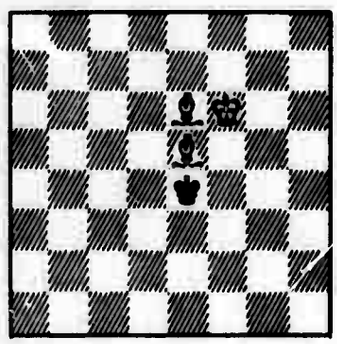


q_1

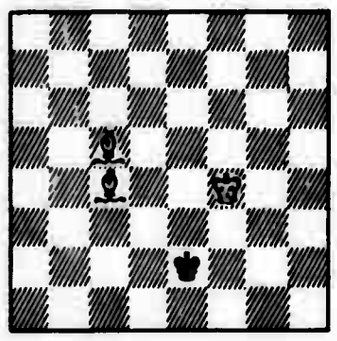
$d_{q_1}(wk, qb) = 3 = d_{q_1}(bk, qb)$. However the white king is unable to prevent the attack on the head bishop since it will be blocked by the black king.



q_2



p_3



q_4

Figure 5.5.

$d(wk_{q_4}, \underline{head}(q_4)) = 3$, but the white king is still able to protect the head bishop. This is because the black king is in check, so it is unable to use the direct attacking path (it needs two moves to attack). The fact that the black king is in check but there is a quadrant, means either the white king is guarding the boundary or the boundary is next to an edge of the board. If the white king is guarding the boundary of the area, we know that it cannot be prevented from protecting the head bishop. If the boundary is next to an edge, white will have no difficulty in forming a smaller quadrant, since we then know that either bishop move will form a smaller quadrant containing a corner. So, if the black king is in check in q , and $d(bk_q, \underline{head}(q)) = 2$, we accept q as long as $d(wk_q, \underline{head}(q)) \leq 3$. The reason for going to all this trouble is that this is a very common occurrence, and if we do not make the exception the program will essentially play from one non-head quadrant to another with only tree search in between. This makes the trees too long. Even so some perfectly safe positions will be rejected.

Let us formalize the conditions discussed in the preceding paragraphs. A function badc is defined to recognize the situations occurring in positions q_2 and p_3 of Figure 5.5. For q , badc is concerned with all positions with $d(bk_q, \underline{head}(q)) = 2$. Thus the case of the black king in check is handled in badc also. We have

$$\underline{safe}(q) \equiv \{(\underline{squad}(q) \text{ is even}) \vee [d(bk_q,\underline{head}(q))=2 \wedge \neg\, \underline{badc}(q)] \vee [d(bk_q,\underline{head}(q))>2 \wedge d(wk,\underline{head}(q))<d(bk,\underline{head}(q))]\} .$$

Note that q with $d(wk_q,\underline{head}(q))=d(bk_q,\underline{head}(q))=1$ will satisfy safe. In p , badc handles all positions with $d(bk_p,\underline{head}(p))=1$. We have

$$\underline{safe}(p) \equiv [(\underline{squad}(p) \text{ is even}) \vee (d(bk_p, \underline{head}(p))=1 \wedge \neg \underline{badc}(p)) \vee (d(bk_p, \underline{head}(p))>1 \wedge d(wk_p, \underline{head}(p))<d(bk_p, \underline{head}(p)))] .$$

Now we can define the recognizer for stage 2:

$$\underline{goodquad}(p) \equiv \{\underline{quad}(p) \wedge \underline{hascorner}(p) \wedge \underline{safe}(p)\}$$

$$\underline{goodquad}(q) \equiv \{\underline{quad}(q) \wedge \neg \underline{spec}(q) \wedge \underline{hascorner}(q) \wedge \underline{safe}(q)\} .$$

goodquad and squad satisfy 3.7 and 3.9.

Stage 3

As explained previously the condition hascorner is used to insure that stage 2 will eventually fit in with Capablanca's method for the end of the game. The example from Capablanca is given in Figure 5.6; position x_1 in Figure 5.3 satisfies the same pattern as Capablanca's position after white move 3. This is the point at which stage 3 should start because now we will use different heuristics. If goodquad(x), then squad(x)>6 indicates stage 2, while squad(x)=5 or 6 gives stage 3. If squad(x)<5 we allow the program to use tree search to arrive at the larger quadrant of stage 3.

Position p_1 in Figure 5.7 is the position in Capablanca after white move 3. Capablanca's strategy for this part of the game is to move the white king up into one of the squares marked X1, X2, or Y, or the square occupied by the black king. For the program, this has been simplified. Only the squares marked X1 and X2 are used as goal squares for the white king. When squad=6, X1 is the goal square. When squad=5, either X1 or X2 is allowed; one of these will be covered by a bishop. Since with squad=5 we have a head quadrant, this is used only as a back-up for squad=6. It is needed

Now we come to two Bishops and King against King.

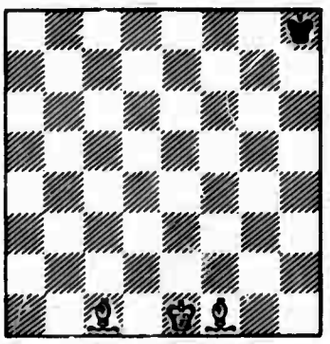
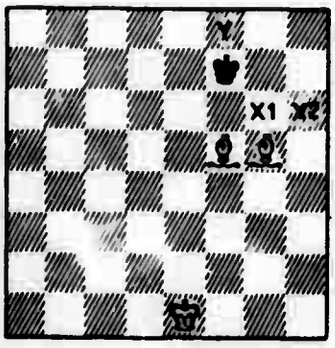


DIAGRAM 22

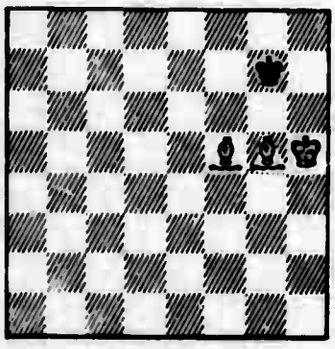
Since the Black King is in the corner, White can play 1 B–Q3, K–Kt2; 2 B–KKt5, K–B2; 3 B–B5, and already the Black King is confined to a few squares. If the Black King, in the original position, had been in the center of the board, or away from the last row, White should have advanced his King, and then, with the aid of his Bishops, restricted the Black King's movements to as few squares as possible.

We might now continue: 3...K–Kt2; 4 K–B2. In this ending the Black King must not only be driven to the edge of the board, but he must also be forced into a corner, and, before a mate can be given, the White King must be brought to the sixth rank and, at the same time, in one of the last two files; in this case either KR6, KKt6, KB7, KB8, and as KR6 and KKT6 are the nearest squares, it is to either of these squares that the King ought to go. 4...K–B2; 5 K–Kt3, K–Kt2; 6 K–R4, K–B2; 7 K–R5, K–Kt2; 8 B–Kt6, K–Ktl; 9 K–R6, K–Bl. White must now mark time and move one of the Bishops, so as to force the Black King to go back; 10 B–R5, K–Ktl; 11 B–K7, K–Rl. Now the White Bishop must take up a position from which it can give check next move along the White diagonal, when the Black King moves back to Ktl. 12 B–KKt4, K–Ktl; 13 B–K6ch, K–Rl; 14 B–B6 mate.

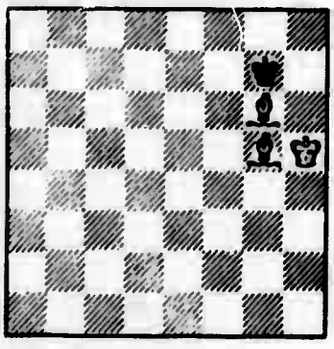
Figure 5.6. Example from Capablance, page 29-30. The program plays almost exactly the same from White move 4 on.



p_1



p_2



q_2

Figure 5.7. Examples of Stage 3.

in a position like p_2 in Figure 5.7; the king's bishop moves to KKt6 which is really a tempo move (position q_2 in Figure 5.7).

The obvious measure for stage 3 is some kind of distance function measuring the number of moves required for the white king to reach the goal square. This function must take account of obstructions (the bishops) and tempo moves. The following function works well. First we define, for position x in stage 3, functions sq1(x) and sq2(x). sq1(x) contains the goal square like X_1 in p_1, and sq2(x) the other goal square X_2. We use as a measure

$$\max (d(wk_x, sq1(x)), d(wk_x, sq2(x))) .$$

This function has a minimum value of 1; it will bring the white king up to sq1 and sq2, but will not select the actual goal square. When a goal square is achieved, we will be in stage 4.

We must consider the problem of satisfying conditions 3.7 and 3.9. If goodquad holds we obviously have no problem, since no new difficulty has been added. Actually goodquad is stronger than needed, since no objection is raised now to moving from squad=5 to squad=6. But there is no particular reason to remove this condition, and it tends to prevent foolish bishop moves.

Stage 4

Once the white king has actually moved into the goal square, the position is in stage 4. (Since st selects the highest stage there is no conflict.) Three factors, recognized by function end2, are used to determine stage 4. One is the position of the white king in a goal square. In addition the black king must be confined to the edge

opposite the white king. This condition will always be satisfied if we are coming from stage 3 and the white king is in the appropriate goal square. If squad=6 and the white king is in sq2, usually the condition is not satisfied. Position q_1 in Figure 5.8 is an example. For the white king as shown, in sq1(q_1), we have stage 4. If the white king were in sq2(q_1)=KR6, the black king would be able to escape from the edge (to KB2), so we would not have stage 4. The third factor is concerned with the distance of the black king and all its legal successors from the corner closest to the white king. Let succ(x) be the set of all successors to the black king in x. Let

succ1(x) = succ(x) $\cup$ bk_x if the black king is not in check in x.

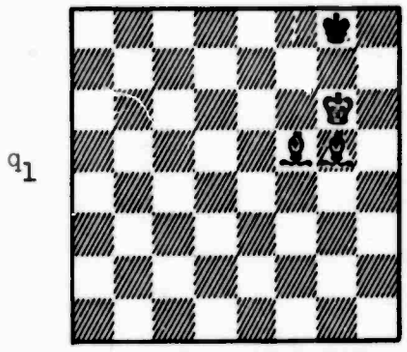
= succ(x) otherwise.

Let c(x) be the corner closest to the white king in x. Then let

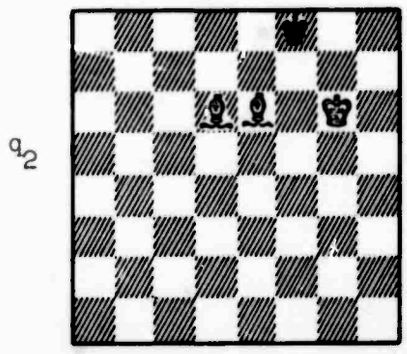
dedge(x) = max({d(r,c(x))|r$\in$succ1(x)})

and dedge(x)$\leq$3 is the condition used for stage 4.

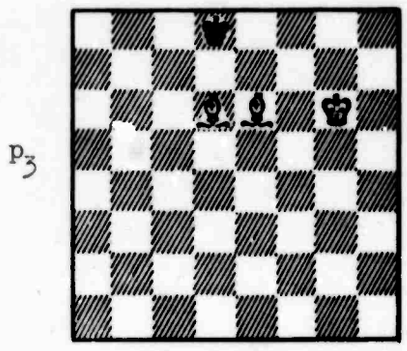
The reason for the choice of three as a limit comes from the fact that this is the highest value which the ordinary entry through stage 3 will satisfy. Sometimes a starting position, like p_3 in Figure 5.8, will have the white king in position and the black king confined to the edge, but farther than three squares from the corner. Either a long tree search or different heuristics would be required to handle such a position if we called it stage 4. This is not worthwhile for such a special case.



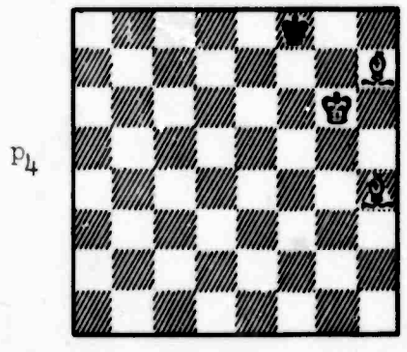
q_1



q_2



p_3



p_4

Figure 5.8. Examples of Stage 4.

The heuristic for this stage is to use the bishops to force the black king into the corner. Checkmate can only be given in or next to a corner in this game. dedge can be used to express this heuristic and is the measure for stage 4.

Again we must worry about satisfying conditions 3.7 and 3.9. The difficulty arises from non-standard entries into stage 4. Consider first q_2 in Figure 5.8. All conditions for stage 4 seem to be satisfied, but when the black king moves to K1 (position p_3 in Figure 5.8) we no longer have a stage 4 position. To avoid such trouble we add condition

$$\underline{\text{dcond}}(q) \equiv \{\underline{\text{dedge}}(q) \leq d(bk_q, c(q)) \vee (\underline{\text{goodquad}}(q) \wedge 5 \leq \underline{\text{squad}}(q) \leq 6)\}.$$

$$\underline{\text{dcond}}(p) \equiv \neg\, \underline{\text{bad4}}(p) \quad .$$

This condition says that the black king is forced to move closer to the corner; we only insist upon this when the entry is not from stage 3. Condition dcond is sufficient to satisfy both rules 3.7 and 3.9, since there is no way in which the black king can force white to abandon stage 4. Since the black king cannot be in check in p, we know that if a bishop is preventing its escape from an edge, that bishop must be bearing on the edge. Unless a bishop is blocked by the white king, as in p_4 in Figure 5.8 (satisfying bad4), white can maintain stage 4. If the white king is preventing the black king from escaping, the bishops have sufficient mobility to keep the advantage.

Formal Definitions of better and worse

Here are the definitions of the stages.

$x \in$ stage 0 $\equiv$ {x is stalemate or x is a position with black to move and black can take a piece in one move}.

$x \in$ stage 1 $\equiv$ {x is not in any other stage}.

$x \in$ stage 2 $\equiv$ {goodquad(x) $\wedge$ squad(x) > 6}.

$x \in$ stage 3 $\equiv$ {goodquad(x) $\wedge$ $5 \leq$ squad(x) ≤ 6 }.

$x \in$ stage 4 $\equiv$ {end2(x) $\wedge$ dcond(x)}.

$x \in$ stage 5 $\equiv$ {x is checkmate}.

The measures are

$m_2(x)$ = squad(x) $\quad x \in$ stage 2

$m_3(x)$ = max(d(wk_x, sq1(x)), d(wk_x, sq2(x))) $\quad x \in$ stage 3

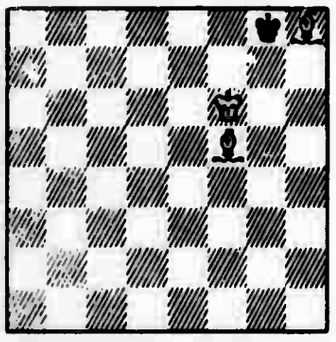
$m_4(x)$ = dedge(x) $\quad x \in$ stage 4

$m_i(x)$ = 0 $\quad i = 0,1,5, \quad x \in$ stage i

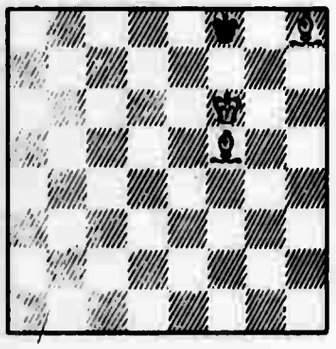
An explanation is needed about the definition of stage 0. There are positions p with white to move which are successors of some $q \in Q$, but $p \notin P$. They are all like position p_2 in Figure 5.9 which is a successor of position q_1 in Figure 5.9. It is not necessary to recognize q_1 as a member of stage 0 however. Since position q_1 is a stage 1 position, and since no position with white to move is in stage 0, q_1 will never be accepted by better. Therefore the program will work correctly with the present definition of stage 0.

Changes to better and worse

Now that we have given the formal definitions of better and worse, we consider what changes are needed to make the program practical. At present a tree of at most depth 3 is required to move from stage 1 to stage 2. This tree is very wide, but since it occurs at most once in a game no changes have been made to stage 1 heuristics.



p_2



q_1

Figure 5.9. Illegal Positions.

In stage 2, very long tree searches may be needed, up to a maximum of depth 8. The worst cases occur in non-head quadrants. Frequently in such positions, tree search to a new non-head quadrant is required because of the difficulty in being certain a head quadrant is safe. For example, in position p_1 in Figure 5.10 a tree of depth 5 is required to force a *better* position; in p_2 the tree has depth 7. We will discuss heuristics for non-head quadrants first.

Obviously we would like to cut down on both the length and breadth of the tree search. Unfortunately it is very difficult to define heuristics to add to *better* which will work in all long trees. In position p_1, the moves wk-QB5 and wk-QB6 are equally good moves, and either would be selected at depth 5. Both moves enable the white king to guard the boundary of the quadrant. The move wk-QB6 satisfies

5.2 $d_q(wk,bk) < d_p(wk,bk) \wedge \underline{dmin}(q) < \underline{dmin}(p)$

where

$\underline{dmin}(x) = \min(d_x(wk,kb), d_x(wk,qb))$,

while wk-QB5 does not. When 5.2 is added to *better* it will cut the tree search in p_1 (starting at level 2 of the original tree) to 4; in p_2 nothing is gained. In many positions, however, considerable reduction in tree search is gained by this heuristic, and the maximum tree depth is cut to 7 (position p_2). 5.2 satisfies 3.7 because *dmin* depends only on the positions of the white pieces. *dmin* also insures that 5.2 will be applied only a finite number of times (no more than five).

Now rule 5.2 will obviously fail if

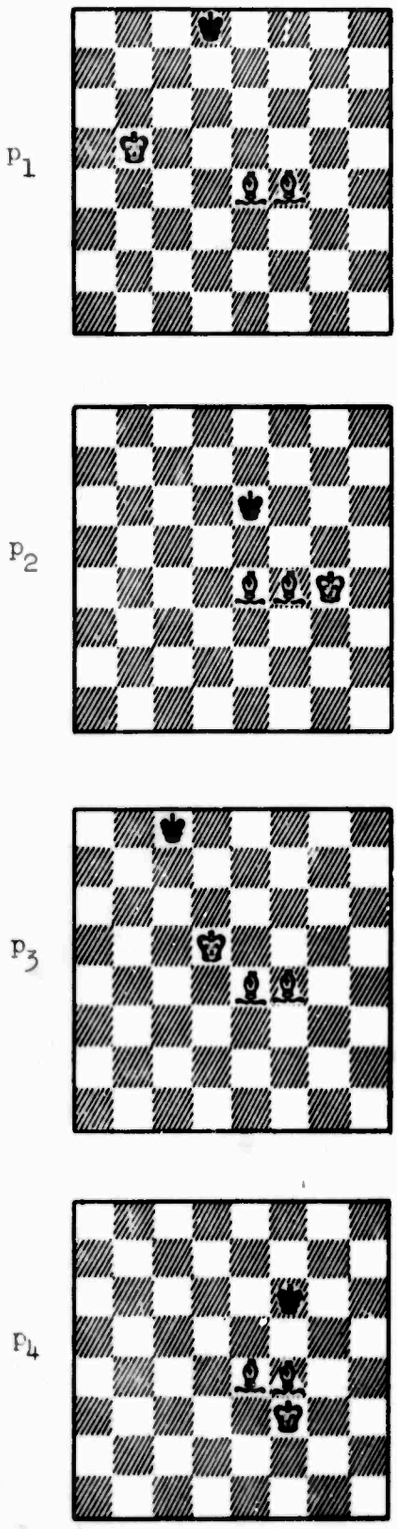


Figure 5.10. Tree Pruning Heuristics for Non-Head Quadrants.

1. $d_p(wk,bk) = 2$, or

2. $\underline{dmin}(p) = 1$.

If these patterns hold in p , we must turn to tree pruning to make the tree manageable. First, all moves leading to positions without quadrants can be eliminated by rejecting q satisfying

$$\underline{badquad}(q) \equiv \{\neg\, \underline{quad}(q) \vee \underline{spec}(q) \vee \neg\, \underline{hascorner}(q) \vee [(\underline{squad}(q) \text{ is odd}) \wedge d(bk_q, \underline{head}(q)) \geq 2 \wedge \underline{badc}(q)\ ]\} .$$

Bishop moves leading to larger quadrants are already eliminated; in addition badquad eliminates some bishop moves leading to smaller quadrants. Few bishop moves are left; these are the ones which hopefully will lead to either a legal head quadrant or a small non-head quadrant in one more move.

badquad applies only to bishop moves; king moves must also be eliminated. First we reject all king moves such that

5.3 $d_q(wk,bk) > d_p(wk,bk)$.

We would also like to reject moves with

$\underline{dmin}(q) > \underline{dmin}(p)$

because although 5.2 is not a measure, since it is a predicate instead of a function with integer values, it would be nice to use it like a measure. However this condition is too strict; in p_3 in Figure 5.10 for example, the move wk-QB6 must be permitted. The condition is changed to

5.4 $\underline{dmin}(q) > 2 \wedge \underline{dmin}(q) > \underline{dmin}(p)$

which works because when $\underline{dmin}(q) > 2$ we have almost no chance of forming a better position with a head quadrant farther down in the tree, so it is much harder to terminate the search.

When $d_p(wk,bk)>2$ it is not always possible to move the white king up to the black king. This is illustrated in p_4 in Figure 5.10. In p_4 the white king is needed on the side of the quadrant toward the center of the board. If he goes there via KKt4, a tree of depth 8 will be required to force *better* positions, while if he goes via K3 the tree terminates at depth 6. In this case we have $d_p(wk,bk)=d_q(wk,bk)$ and *dmin*$(q)=1$. We define

5.5 $d_p(wk,bk)=d_q(wk,bk) \wedge d_p(wk,bk)>2 \wedge$ *dmin*$(q)>1$

as our final heuristic for rejecting king moves in non-head quadrants.

In head quadrants there is usually less difficulty in forcing a *better* position since a non-head quadrant is automatically safe. In general the tree searches are not as long as for head quadrants before the addition of 5.2; a depth less than four is average. Position p_1 in Figure 5.11 is an example; this position may occur after the tree search from position p_1 in Figure 5.10. A tree of depth 2 is required and the first move should be any white king move but wk-K5 or wk-K6. The heuristic added to *better* for non-head quadrants does not apply and this is true in general for head quadrants. Since the trees are of manageable length no changes have been made to *better*.

Slightly different heuristics are used for tree pruning for head quadrants than for non-head quadrants. *badquad* is replaced by the stronger condition that only legal stage 2 positions are permitted for q . This rule eliminates king as well as bishop moves. Other king moves are rejected by

5.6 *dmin*$(q)>1 \wedge (d_q(wk,bk)>d_p(wk,bk)$

$\vee [d_q(wk,bk)=d_p(wk,bk) \wedge d_p(wk,bk)>2])$.

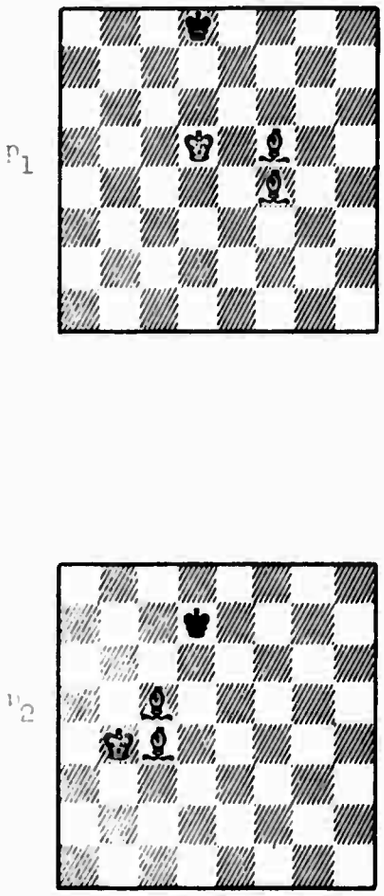


Figure 5.11. Examples of Head Quadrants.

In head quadrants it may actually be necessary to move the white king away from the black king. This is shown in position p_2 in Figure 5.11. It is essential to move the white king to QB3 at this point; the move is similar to the one made in p_4 in Figure 5.10. A tree of depth 5 is needed from p_2. The correct move is permitted since $\underline{dmin}(q)=1$.

Summing up the additional heuristics in stage 2, we add to *better*

5.7 $\underline{st}(p)=\underline{st}(q)=2 \wedge (\underline{squad}(p)$ is even$) \wedge d_q(wk,bk)<d_p(wk,bk)$
$\wedge \underline{dmin}(q)<\underline{dmin}(p)$.

We add to *worse*

5.8 $\underline{st}(p)=2 \wedge [(\underline{st}(q)=1 \wedge [(\underline{squad}(p)$ is odd$) \vee \underline{badquad}(q)])$
$\vee (\underline{st}(q) 2 \wedge m_2(p)=m_2(q) \wedge \underline{cutk}(p,q))]$

where

$\underline{cutk}(p,q) \equiv \{[(\underline{squad}(q)$ is even$) \wedge (d_q(wk,bk)>d_p(wk,bk)$
$\vee [d_q(wk,bk)=d_p(wk,bk) \wedge \underline{dmin}(q)>1$
$\wedge d_p(wk,bk)>2]$
$\vee [\underline{dmin}(q)>2 \wedge \underline{dmin}(q)>\underline{dmin}(p)])] \vee$
$[(\underline{squad}(p)$ is odd$) \wedge \underline{dmin}(q)>1 \wedge$
$(d_q(wk,bk)>d_p(wk,bk)$
$\vee [d_p(wk,bk)=d_q(wk,bk) \wedge d_p(wk,bk)>2])]\}$

combines the king move heuristics 5.3, 5.4, 5.5, and 5.6.

In stage 3, the formal definitions work very well. Considerable tree pruning can be gained by adding to *worse*

5.9 $\underline{st}(p)=3 \wedge \underline{st}(q)<3$,

which will not eliminate all paths to *better* positions. The tree searches have a maximum length of 3, and with the addition of 5.8, a width of no more than three moves at any level.

In stage 4 we are also doing fine as far as tree length is concerned since the tree will only have a depth of 2. We add

5.10 $\underline{st}(p)=4 \wedge \underline{st}(q)<4 \wedge \neg\, \underline{end2}(q)$

to worse; even with 5.9 the tree is quite wide but this is not serious since it is so short.

Combining 5.7, 5.8, 5.9 and 5.10 with the formal definitions of better and worse we have

$$\begin{aligned}\underline{better}(p,q) \equiv \{&\underline{st}(q)>\underline{st}(p) \vee [\underline{st}(q)=\underline{st}(p) \wedge m_{\underline{st}(p)}(q)<m_{\underline{st}(p)}(p)]\\ &\vee [\underline{st}(p)=\underline{st}(q)=2 \wedge (\underline{squad}(p) \text{ is even}) \wedge\\ &\quad d_q(wk,bk)<d_p(wk,bk) \wedge \underline{dmin}(q)<\underline{dmin}(p)]\} .\end{aligned}$$

$$\begin{aligned}\underline{worse}(p,q) \equiv \{&\underline{st}(q)=0 \vee [\underline{st}(q)=\underline{st}(p) \wedge m_{\underline{st}(p)}(p)<m_{\underline{st}(p)}(q)]\\ &\vee [\underline{st}(p)=2 \wedge\\ &\quad ([\underline{st}(q)=1 \wedge ((\underline{squad}(p) \text{ is odd}) \vee \underline{badquad}(q))]\\ &\quad \vee [\underline{st}(q)=2 \wedge m_2(p)=m_2(q) \wedge \underline{cutk}(p,q)])]\\ &\vee [\underline{st}(p)=3 \wedge \underline{st}(q)<3]\\ &\vee [\underline{st}(p)=4 \wedge \underline{st}(q)<4 \wedge \neg\, \underline{end2}(q)]\} .\end{aligned}$$

These are the functions used by the program.

Examples of Program Play

Our first example will illustrate how the program plays the last part of the game. We will start with the position occurring after black move 3 in Capablanca's example (Figure 5.6). This position is the same as p_1 in Figure 5.7 except that the black king is in KKt2. The program would not make the same first moves as are given in Capablanca because a search of depth 3 has been made while the program will use a depth 2 tree. We have

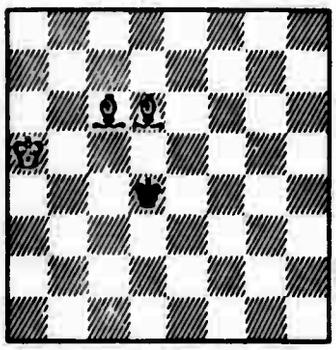
4.	wk-Q2		This move gives $m_3(q)<m_3(p)$, but it is not as good as the book move wk-KB2.
		bk-KB2	
5.	wk-K3	bk-KKt2	
6.	wk-KB4	bk-KB2	
7.	wk-KKt4		We have lost one move.
		bk-KKt1	
8.	wk-KR5		Moves 7 and 8 are found by a tree search of depth 2.
		bk-KKt2	
9.	kb-KKt6	bk-KKt1	
10.	wk-KR6		Again by a tree search of depth 2.
		bk-KB1	
11.	qb-KB6	bk-KKt1	
12.	qb-K7		Again a tree of depth 2. The program's move 11 is just as good as the book's move 10 (it is a tempo move).
		bk-KR1	
13.	kb-KB5	bk-KKt1	
14.	kb-K6 ch.	bk-KR1	
15.	qb-KB6 mate.		

This example shows that the program plays the last part of the game very well. Its only mistake is move 4 and this is not serious.

Our next example is taken from Fine (Figure 5.1). Our starting position, p_1 in Figure 5.12, occurs after black move 2. The program will make different initial moves than the book because of the order of move generation (position p_2 in Figure 5.12 would result). We have

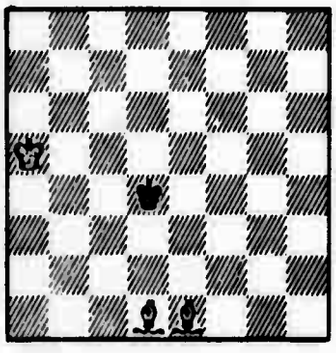
3.	wk-QKt6		This move is not nearly as good as the book move or wk-QKt5. Move generation is at fault again.
		bk-Q6	
4.	kb-Q5		We are now playing differently from the book.
		bk-K6	
5.	wk-QB7	bk-Q5	
6.	wk-QB6	bk-K6	
7.	wk-Q7		We need the king on the other side of the quadrant.
		bk-Q5	
8.	wk-K6	bk-K6	
9.	wk-K5	bk-KB7	
10.	wk-KB4	bk-KKt8	
11.	wk-KKt3		Condition dcond prevents the program from accepting the position at this point (q_3 in Figure 5.12) as better.
		bk-KB8	
12.	qb-QB5		Moves 5-12 are found by a tree search of depth 8. The black moves are on the longest branch.

p_1



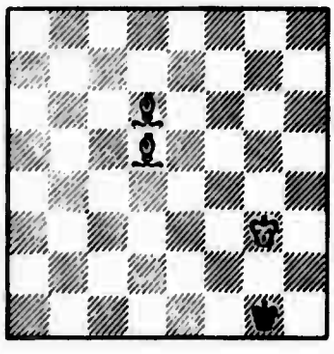
Position p_1 is the start of the second example of program play.

p_2



The program arrives at p_2 after two moves from the initial position in Figure 5.1.

q_3



q_3 occurs after program move 11; $\underline{\text{end2}}(q_3)$ is true but $\underline{\text{dcond}}(q_3)$ is false, which prevents the program from accepting q_3 as <u>better</u>.

Figure 5.12. Examples of Program Play.

		bk-K7	
13.	wk-KB4	bk-Q6	
14.	wk-KB3	bk-QB6	
15.	wk-K4	bk-Q7	
16.	kb-QB4		Moves 13 through 16 form a branch of length 4 in a tree of depth 6.
		bk-K8	
17.	wk-K3	bk-Q8	
18.	qb-QKt4		We are now in stage 3.
		bk-QB7	
19.	wk-Q4	bk-QB8	
20.	wk-QB3	bk-Q8	
21.	wk-QKt3		

Move 21 gives a stage 4 position, and the play from this point on is essentially identical to the first example. Five more moves are required to mate. This means that the program uses 24 moves to reach checkmate from p_1 in Figure 5.12, while the book uses 22. Therefore the program is playing quite well in spite of the interference caused by bad move generation. The moves selected for black vary from ones which present white with maximum difficulty (for example, black moves 4 through 11) to medium difficulty (black moves 12 through 15). Similar kinds of black moves are given in the book. The program would require about 28 moves to reach checkmate from p_2, so for the entire example, it uses six more moves than the book.

The only place where the program is likely to have difficulty in this example is with the tree of depth 8 (moves 5 through 12).

Fortunately this tree is very narrow. Since the position at the beginning of the tree has a head quadrant, most black moves allow white to form a <u>better</u> position immediately. There is one other main branch in the tree (wk-QB5); this branch would terminate at depth 9. This tree provides an illustration of the necessity of allowing the white king to move away from the black king. Generally trees from head quadrants are short (for example, moves 17 and 18); the one exception occurs when the presence of the white king is required on the other side of the quadrant, as in this tree.

One last example is given to illustrate some remarks made about non-head quadrants. We begin with position p_4 in Figure 5.10.

1. wk-K3 — The white king is taking the shortest route to the other side of the quadrant.

bk-K3

2. wk-Q4 bk-Q2
3. wk-Q5 bk-QB1
4. wk-QB6 bk-Q1
5. kb-KB5 — We have not yet reached a <u>better</u> position because the white king is too far away from the head bishop.

bk-K2

6. wk-Q5

Now we have reached a <u>better</u> position. At move 6, qb-KKt5 would give a smaller non-head quadrant, but unfortunately this move was not generated soon enough.

Evaluation of Program Play

The program is playing adequately, and the comments made at the end of Chapter 4 can be applied to this game also. We merely note that a second-best move in this game hurts the program more. Since the game is harder, more precision is required for good play.

The program play is very close to book play in the last part of the game. This is not true in the first part. However, the method used in the first part was suggested by the book and works well.

BLANK PAGE

CHAPTER 6

BISHOP, KNIGHT AND KING AGAINST KING

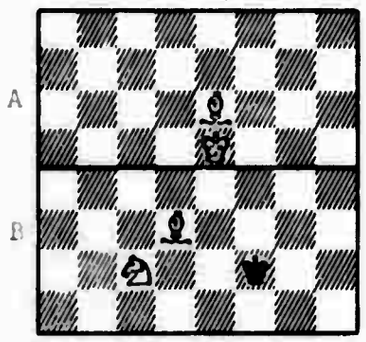
This end game is one of the most difficult of the classical endings. When it is discussed in the chess books, it is broken into two main parts. The first part of the game consists in forcing the black king to an edge. Since the mate can only be given in (or next to) a corner of the same color as the bishop (the black corner in this discussion since we will assume that white has the queen's bishop), we expect to finish the first part with the black king in the corner of opposite color to the bishop (the white corner). Then the second part consists in forcing the black king down the edge to the corner where mate can be given.

While the method of play used by the program in the second part of the game agrees exactly with the books, in the first part we are forced to provide our own heuristics. There are two reasons for this. First, the books only give a limited example of this part of the game; the program must be able to handle all black king moves, not just those that are most likely. And although books do make some attempt to explain how to play, the procedures described are too local in nature to be used directly. Figure 6.1 is taken from Fine [1944]; the two patterns described are quite powerful, and in his example very conveniently the white pieces are in a position to make constructive use of them. However these patterns are useful in general only when embedded in some global heuristic.

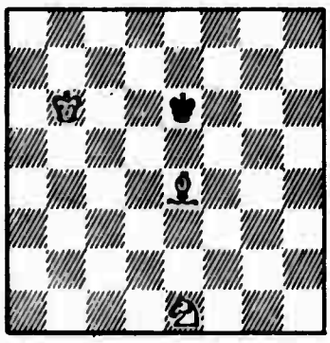
In order to drive the enemy King back to the edge of the board White must make use of two typical positions (see diagram No. 5). In the first - 5A - all the points leading towards the center are inaccessible to the Black King and he cannot maintain the status quo; he must retreat. In the second - 5B - the two pieces are cooperating beautifully. Black's King can do nothing better than mark time and as soon as the White King comes up he will have to give way. The important feature in No. 5B is that the two pieces are diagonally adjacent to one another, for it is because of this fact that they cover so many squares.

Starting from some arbitrary position such as No. 6 the most effective continuation would 1 Kt–B3 (No. 5B), K–Q3; 2 B–B6 (No. 5A), K–K3; 3 K–B5, K–K2; 4 K–Q5, K–B1. Black is well advised to go to the

No. 5. Driving the Black King Back.



No. 6



wrong corner, for that is the only way in which he can hold out for any appreciable time. 5 K–K6, K–Kt2; 6 Kt–K5, K–B1; 7 K–B6, K–Kt1; 8 Kt–Kt6, K–R2; 9 B–Q5, K–R3; 10 B–Kt8 and now we have position No. 4 since the fact that Black will be chased along the file rather than along the rank makes no difference.

No. 4

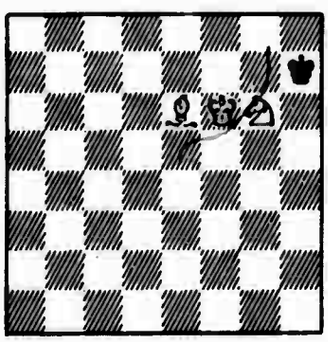


Figure 6.1. Example from Fine, pages 18-20.

Stage 0

Stage 0 as usual contains the various illegal positions which in this end game comprise quite a large class. It contains positions in which the black king can take a piece in two moves as well as the usual loss in one move. Since immediate loss or stalemate is obvious we concentrate on describing the other kind of stage 0 position.

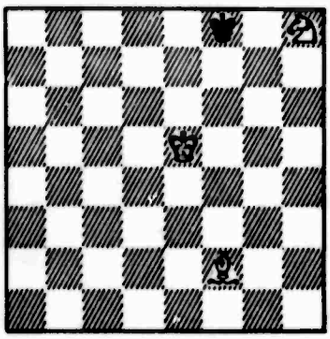
In order to be sure that we discover all illegal positions we consider how such positions might occur. First, suppose the black king can attack only one of the bishop and knight. Since the bishop has so much mobility, it will be able to escape the black king unless its path is blocked by the knight. Therefore the knight is also under attack, and this case will be discussed later. The knight does not have as much mobility as the bishop and in fact is open to attack if it is in a corner. Examples are shown in q_1 and q_2 of Figure 6.2. To avoid having to recognize positions like q_1 and q_2 (and distinguish them from similar positions in which white is able to protect the knight) we assign all positions in which the knight is in the corner to stage 1, which in this game proceeds the catch-all stage. This insures that the program will move the knight out of a corner if it is in one in a starting position, and will never accept such a position as <u>better</u>.

It is also possible for the black king to attack the knight and bishop at the same time. The attack must come in one move or white will be able to avoid it. We have

$$d_q(kt,qb)\leq 2 \wedge d_q(bk,qb)\leq 2 \wedge d_q(bk,kt)\leq 2 .$$

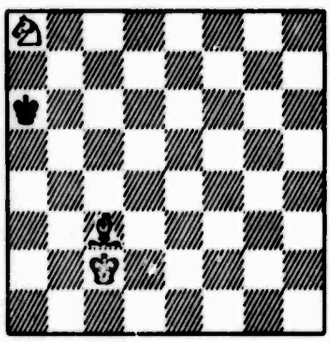
(The reader should refer to Appendix A for definitions of basic functions and notation.) We also assume that neither the knight nor the bishop is

q_1



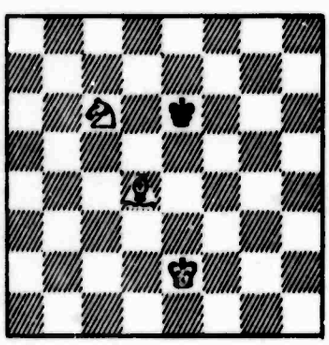
After bk–KKt2, white will be unable to avoid losing the knight.

q_2



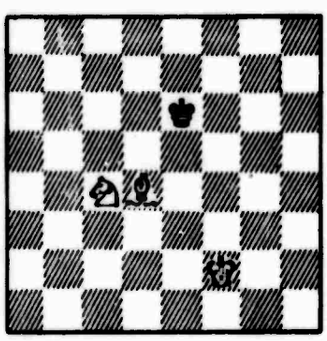
After bk–QKt2, white will be unable to avoid losing the knight.

q_3



After bk–Q4, white will be unable to avoid loss of a piece.

q_4



After bk–Q4, white will be unable to avoid loss of a piece.

Figure 6.2. Examples of positions in which black can take a piece in two moves. Positions q_3 and q_4 are in stage 0, but q_1 and q_2 are in stage 1.

susceptible to being taken immediately. If $d_q(qb,kt)=2$, there is only one configuration of knight and bishop which permits such an attack. It is illustrated in q_3 in Figure 6.2. If the black king instead were on K4, K5 or QB5 he could also move to attack both pieces. In q_3 the knight is on a white square, and consequently is bearing on a black square. This means that it is not able to move to protect the bishop, and also the bishop cannot possibly move to protect it. Since the black king will threaten both the bishop and the knight, it is not possible to simply move a piece out of danger. Therefore, the white king is white's only means of defense. If the white king is next to either piece the loss can be avoided. Also, if the white king can come to the aid of the knight no loss will occur since the knight protects the bishop. So we will lose a piece if

$$d_q(wk,qb)>1 \wedge d_q(wk,kt)>2 .$$

If $d_q(kt,qb)=1$ we have several cases to consider. First we have positions like q_4 in Figure 6.2 in which the knight is on a white square. The black king could also be on Q3 or QB3 and be able to move to attack the pieces. Such a position is similar to the previous case, but in q_4 the knight does not protect the bishop, so the white king must be able to move to protect both pieces if loss is to be avoided. Therefore

$$d_q(wk,qb)>2 \vee d_q(wk,kt)>2$$

implies a piece will certainly be lost. In addition, even if this condition is not satisfied white may still lose a piece since the move black makes to attack may block the white protecting move. This would happen in q_4 if the white king were on QB6.

Positions q_1 and q_2 of Figure 6.3 are examples of $d_q(kt,qb)=1$ with the knight on a black square. In such positions the bishop is protecting the knight. If the bishop were not on an edge, it would be able to retreat from the black attack and continue to protect the knight. If in q_1 or q_2 we had $d_q(wk,qb)=1$, then the white king would prevent the black king from moving into the attacking square. Also not all the squares two away from the bishop are forbidden to the black king; for example in q_1 only from squares Q6, K6, and KB6 can the black king force the loss of a white piece. As usual, we do not worry about $d_q(wk,qb)>2$ since we will handle that through stage 1. All of the various cases of positions two black moves away from the loss of a piece will be recognized by function badpos(q).

In the positions shown in Figure 6.2, and also in q_1 and q_2 in Figure 6.3, the black king causes difficulty for white by attacking pieces. It is also possible for black to combine a threat of stalemate with an attack on a piece. Position q_3 in Figure 6.3 is an example. There is no danger that this position would be chosen by better in some later stage. Therefore it is not necessary to recognize it.

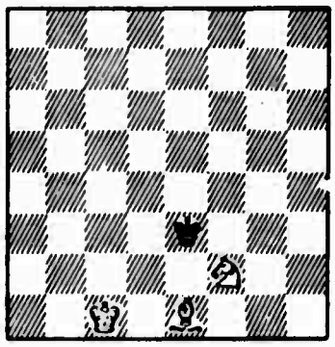
Stage 1

As mentioned during the discussion of stage 0, stage 1 is inserted before the catch-all stage because this is a way of using simple tests to avoid a lot of pattern recognition. Stage 1 contains all positions with the knight in a corner and also all positions where

$$d_p(wk,qb)>d_p(bk,qb)+1$$

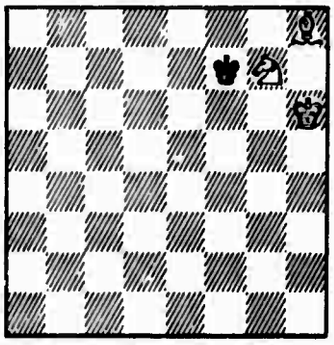
$$d_q(wk,qb)>d_q(bk,qb)$$

q_1



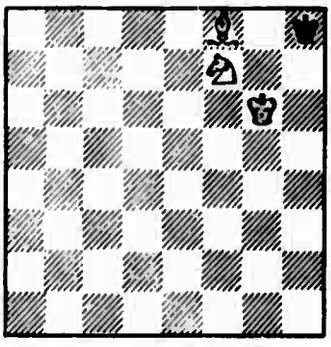
If the black king moves to K7, white will lose either the knight or the bishop.

q_2



If the black king moves to KKt1, then white will lose the bishop.

q_3



After the black king moves to KKt1, white will either lose the bishop or give a stalemate.

Figure 6.3. More examples of positions in which black can force a draw in two moves. Position q_1 and q_2 are in stage 0; position q_3 would never be accepted as <u>better</u>, so we need not worry about recognizing it.

provided such positions are not already in stage 0. All of these conditions are recognized by function stage1.

There are many positions p which are not in stage 1 but all of whose successors are. In such a p the black king is attacking the knight and white must move the knight away to protect it. It may then happen that the black king is closer to the bishop than the white king is, giving a q in stage 1. We will not worry about recognizing either p or a q which preceeds this p because the strategy in the later stages is equipped to handle such a p . Therefore p remains in the stage it should be in (generally stage 2), and we do not break rule 3.7 although we do violate rule 3.9.

Stage 2

Since stage 1 has other uses, stage 2 is the catch-all stage whose presence is recognized by the absence of all other stages. Position 6 in Figure 3.1 is in stage 2. A measure will be given for this stage. This measure is based on the statement in Capablanca [1935] which says that we should begin this end game "by advancing the king to the center of the board".[3] One result of following this rule is that the program will move wk-QB6 or wk-QB5 in position 6 in Figure 6.1. There are four squares in the center of the board; they are Q4, Q5, K4, K5. So we define as our measure for stage 2 the function

$$\underline{\text{dcent}}(x) = \min_{sq \in S} (d(wk_x, sq)), \quad S = \{Q4, Q5, K4, K5\}$$

There is no difficulty in showing that rule 3.7 holds for dcent, since this function depends only on the position of the white pieces. We do expect to break rule 3.9 occasionally by having all successors to

3. Page 109.

some p be in stage 1. To use dcent as a measure in worse, we must be sure that it is never necessary to move the white king away from the center of the board. Although the black king can move into a position p which would be in stage 0 if it were a position with black to move, white will always be able to avoid stage 0 without moving the white king away from the center. Since this p is in stage 2, we know that the knight is not in a corner, and $d_p(wk,qb) \leq d_p(bk,qb)+1$. An example of such a position is given in p_1 in Figure 6.4. We will avoid the loss of a piece by moving the white king to K4 and then the knight to KB3. p_1 is representative of such stage 2 positions; if it is not possible to move the knight immediately, there will be a king move which will enable us to move the knight and protect the bishop on the next move. This king move will generally give a position q in stage 1; the point is, it is not necessary to allow the white king to move away from the center (such a move would probably give a stage 2 position). Therefore we can use dcent as a measure.

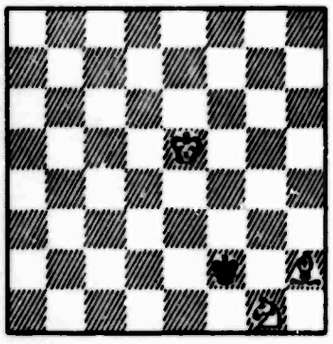
Stage 3

The positions in stage 3 have a definite pattern dependent on recognizing that the black king is contained in a certain area of the board. A size s can be assigned to this area and as usual we will attempt to shrink s . However s cannot be used as a measure. It can be used like a measure in better; that is

$$st(p)=st(q)=3 \wedge s(p)>s(q)$$

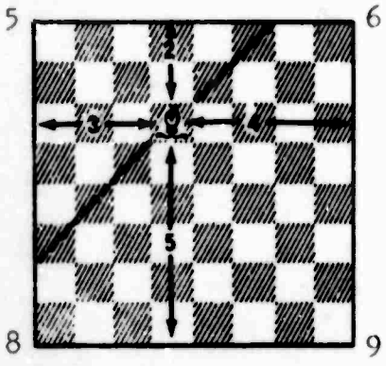
will mean that q is better than p . But s cannot be used in worse because in a few cases this part of worse

p_1



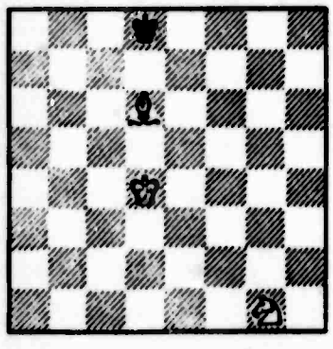
p_1 in is stage 2, and dcent(p_1) = 0. Therefore wk–KB4, the only move giving a stage 2 position, will be rejected since dcent(q) = 1. However, wk–K4, which gives a stage 1 position, will permit kt–KB3 on the next move, thus avoiding the loss of a piece.

x_1



One corner of the board corresponds to the right angle of the area triangle. The size of that area is marked at the corner.

x_2



The black king is inside areas of size 5 and 6.

Figure 6.4.

$\underline{st}(p)=\underline{st}(q)=3 \wedge s(p)<s(q)$

would eliminate the only move(s) which the program must make to proceed correctly. When this happens, it is because the pattern recognition on q is not sufficient to define the real value of s(q) . Since such violations occur infrequently it is of course possible to add pattern recognition to assign the proper value to the offending q . However this approach is not taken. First of all, the pattern recognition would have to be extremely detailed to define s(q) correctly and it is not worthwhile to do all this analysis. As long as s(q) is never smaller than it should be we can be sure the program will not accept q for the wrong reasons. Also s satisfies rule 3.7; once a q has been accepted, we know that for any p which follows from it by one black move, either p is in a higher stage than q or if p is in stage 3 then $s(p)\underline{\leq}s(q)$. Therefore the program will be able to proceed consistently even if s(p) is larger than it should be.

Second is the fact that throughout stage 3 we are liable to break rule 3.9, generally by having all successors to a p in stage 3 in a lower stage, and when <u>worse</u> is occasionally incorrect this is only a special case of the overall problem. As explained in Chapter 3, rule 3.9 is useful but not necessary, and in this end game the amount of pattern recognition required to satisfy rule 3.9 is not worthwhile.

Briefly, the reason for the violation of rule 3.9 is the following. In the preceding games the black king could escape from an area in at most one way, but in this game the black king will be able to escape from the area defined for stage 3 in many different ways. Some of these will force a larger area and so must be prevented, but the majority will

put black in a poor position from which he must retreat or white will be able to ultimately "confine" the black king to a smaller area of the board. "Confine" is put in quotes here because of course the same kind of escape may be available to black in the smaller area. White should take advantage of such moves; the problem is that the smaller area may not be recognized right away, and in the meantime we may break rule 3.9.

First let us see what kind of area we can use to define stage 3. We must partition the board globally or we might not be consistent in our evaluation of successive positions (satisfy rule 3.7). Therefore, the bishop must be the primary piece involved in defining the area, since it is the only white piece which can hold a line through the entire board. In this game we will deal with halves rather than quarters of the board. For any bishop position there are two diagonals, and each diagonal defines areas on both its sides. Therefore there are four different areas to consider. (If the bishop is in a corner there are only three.)

We assign a size to each area in a very simple way. An area is a right triangle in shape with the hypotenense the bishop diagonal. It may be necessary to extend the board to complete the triangle. The other two sides are edges of the board; call them edge1(x) and edge2(x). Then the size of the area is

6.1 $de(qb_x, edge1(x)) + de(qb_x, edge2(x))$,

for de as defined in Appendix A. x_1 in Figure 6.4 provides an illustration of areas. For the bishop diagonal as drawn, the area above the diagonal has size 5, and the area below has size 9. The other diagonal

defines areas of sizes 6 and 8. In stage 3 we are only interested in areas of size less than or equal to 6.

So far we have only discussed how to assign a size to an area. We have not said which area is used to represent a position. Making this decision is a complicated procedure. As explained before, the black king will have many points of escape from an area in this game. We do not want to block all escapes but only those which would force a larger area. However we must satisfy rule 3.7. To accomplish this we insist that an area in q holds if all the successors of the black king are in it, while in p we recognize the area if it contains the black king. Then we can be sure that after one black move the program will be able to see the same area which it used as the basis for accepting q .

Now suppose the black king is placed on the board. The black king is necessarily inside one area, and sometimes inside two. For example, in x_2 in Figure 6.4, the black king is in an area of size 5 and an area of size 6. We must decide which of these areas to use. Obviously we want (1) to assign the smaller area if possible and (2) to be sure the black king cannot escape from the assigned area into a larger one. We have already stated that the black king cannot escape in one move in q ; however it may be able to escape in two moves in q and consequently in one move in p . Since it is difficult to calculate whether the knight can be brought into position to block an escape, we rely mainly on the white king.

The way we decide about an area is as follows. First we use the position of the black king relative to the bishop to propose an area.

This condition is different for positions p and q . To do this we define a function which selects areas:

area(x,C) = (the area on the board whose right angle is corner C).

For any area a , c(a) produces the corner which is the right angle of a . Now we define

dc(sq,a)=de(sq,fileedge(c(a)))+de(sq,rankedge(c(a)))

where sq is some square on the board, and fileedge and rankedge produce the appropriate rank and file containing c(a) . Then

size(x,a)=dc(qb_x,a)

is the correct definition of the size of the area and agrees with 6.1. This function dc is basic to the kind of area with which we are concerned because it has the same value for any square on a diagonal parallel to the boundary of the area. We can also use it to determine where a square sq is with respect to an area a by

location(x,sq,a)=size(x,a)-dc(sq,a) .

If location(x,sq,a) is positive then sq is inside a ; if it is zero sq is on the boundary of a and if it is negative sq is outside a . location is also used to tell how far the diagonal containing the square is from the boundary.

Let succ(x) be the set of squares to which the black king can legally move in x . Now we can define for area a

inside(q,a) ≡ [location(bk_q,a)≥ 0 ∧

∀r(r∈succ(q) ⊃ location(r,a)>0)]

inside(p,a) ≡ location(bk_p,a)>0

The definition of inside for q insures that the black king must move inside the area, and this will then be recognized by inside for p .

Once an area has satisfied *inside* we are ready to make further tests on the positions of the white king and bishop. First we insist that the bishop be placed toward the center of the boundary of the area. Recall that any bishop position on the boundary of a given area will produce the same value for *size*. The condition is

$$\underline{bpos}(x,a) \equiv [d(qb_x, c(a)) \leq (\underline{size}(x,a)-2)] \ .$$

The reason for this condition is that when the bishop is placed toward the center of the boundary it is easier for white to form a smaller area and also to control the black king if he tries to escape. If $\underline{size}(x,a)<4$, no squares would satisfy *bpos* and in fact areas of size less than 4 are not used.

If the bishop is in an acceptable position, the program will examine the position of the white king and its relationship to the bishop and black king. First the white king must be outside the area, i.e.,

$$\underline{location}(x, wk_x, a)<0 \ .$$

Also we always have

6.2 $d_x(wk, qb) \leq 2$

and the white king must be close enough to the bishop to protect it; otherwise we would be in stage 1. The final condition on the white king position is

$$\underline{kpos}(x,a) \equiv (d(wk_x, c(a)) < \underline{size}(x,a))$$

which says that the white king must be fairly centrally located. These conditions are illustrated in Figure 6.5. In x_4 and x_5 all possible squares satisfying *kpos* and *location* will also satisfy 6.2, but some squares may be eliminated in x_6 . Summing up all the conditions stated so far, we have

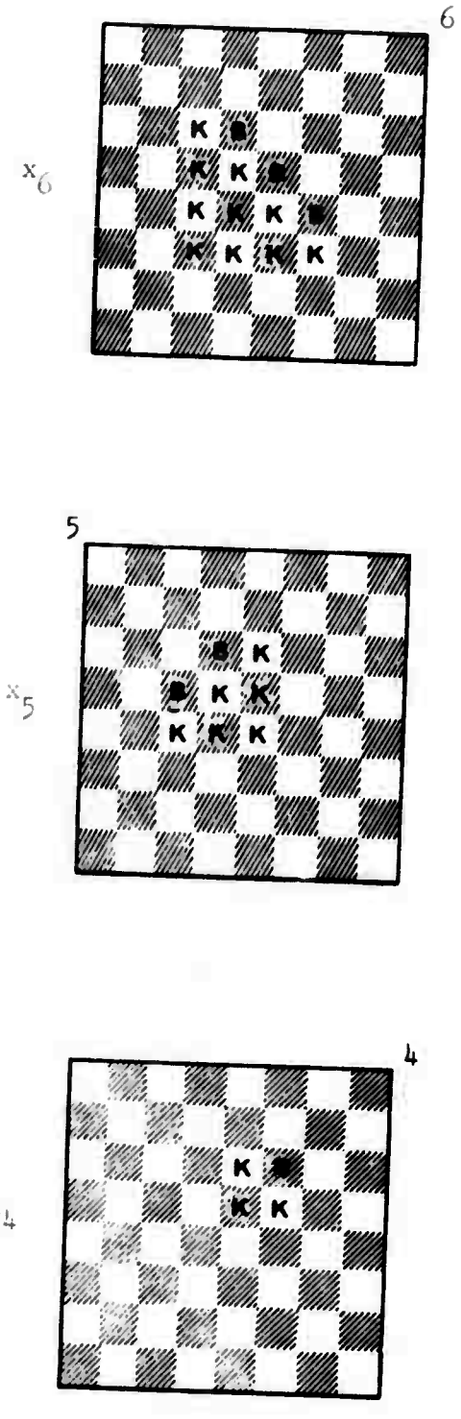


Figure 6.5. Legal squares for the bishop are marked B ; for the white king they are marked K.

$$\underline{\text{safe}}(x,a) \equiv \{\underline{\text{inside}}(x,a) \wedge \underline{\text{location}}(x,wk_x,a)<0 \wedge$$
$$\underline{\text{bpos}}(x,a) \wedge \underline{\text{kpos}}(x,a) \wedge d_x(wk,bb)\leq 2\} .$$

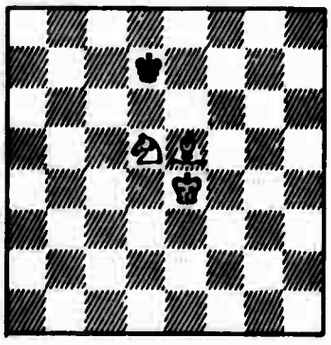
These conditions are correct as far as they go, but we have not paid any attention to the knight. Actually we want to use the knight to help force a smaller area, but when the knight is not being used it possibly will be a hindrance. There are three ways in which the knight can interfere: it can block the white king or bishop, or it can force white to lose a move by being open to black attack. Examples of the three different types of interference are shown in Figure 6.6. All of these examples could arise as the result of one black move from a position q which has an area satisfying safe. None of the kinds of knight interference shown in Figure 6.6 is bad since white can always either maintain the same area or find a smaller one very shortly. Therefore there is no reason to forbid the kind of interference shown in these three positions.

We do want to forbid certain kinds of knight interference however. We use the same guiding principal for eliminating knight positions as we have all along; we cannot allow the black king to force a larger area. There are two kinds of bad knight positions. These are shown in q_1 and q_2 in Figure 6.7. In both cases the black king will be able to attack the knight in one move and thus escape toward the center of the board. Even so white has no trouble controlling the escape when the bishop satisfies

$$d(qb_x,c(a))\leq \underline{\text{size}}(x,a)-3 ,$$

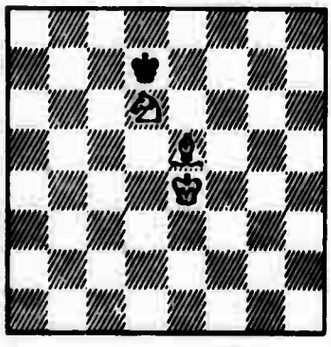
because this insures that the white king will be able to block the escape (since in q it is protecting the bishop). The patterns shown

p_1



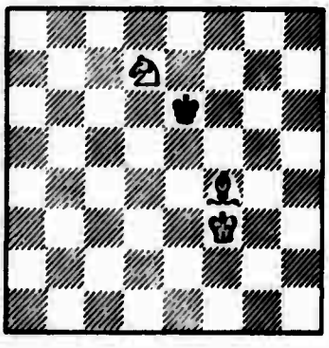
The knight is blocking the white king.

p_2



The knight is blocking the bishop.

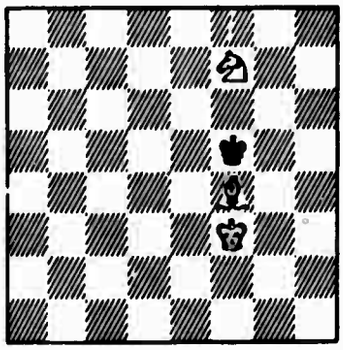
p_3



The knight is being attacked.

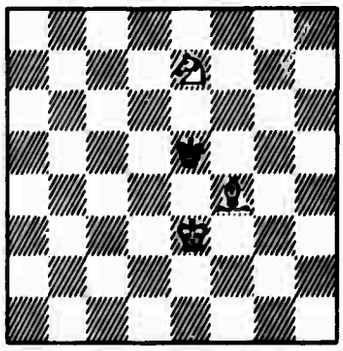
Figure 6.6. Examples of Knight Interference.

q_1



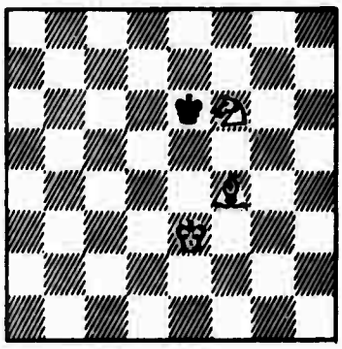
If the black king moves to K_3 it will then be able to escape.

q_2



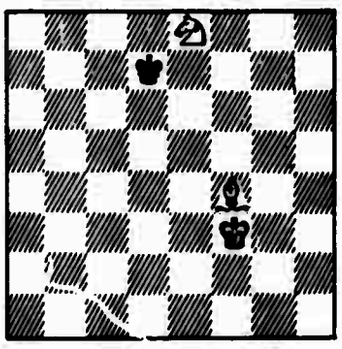
Black can escape by first moving to K_3.

p_3



This position is essentially the same as the result of bk–K_3 in q_1.

p_4



If the white king were in K_3 the escape could be blocked

Figure 6.7. Forbidden Knight Interference.

in q_1 and q_2 are the only bad ones in q (with minor variations) and are recognized by

$$\underline{\text{badkt}}(q,a) \equiv \{d_q(kt,qb)=3 \wedge d_q(bk,qb)=1 \wedge d_q(bk,kt)=2$$
$$d(qb_q, c(a)) \geq \underline{\text{size}}(q,a)-3 \wedge \underline{\text{location}}(q,kt_q,a)=2$$
$$\vee \{\underline{\text{location}}(q,bk_q,a)=0 \wedge \underline{\text{location}}(q,kt_q,a)=3)\} .$$

More pattern recognition is required in p because we must be prepared for bad initial positions as well as results of one move from a q satisfying badkt. Position p_3 and p_4 are examples. Both these positions cannot possibly have come in one black move from a position in which the area of the appropriate size was recognized. We have

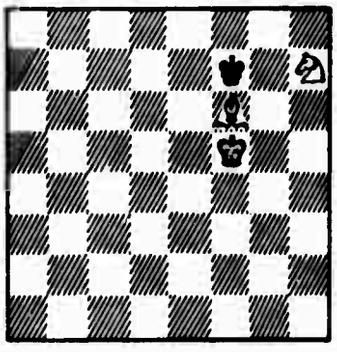
$$\underline{\text{badkt}}(p,a) \equiv \{\underline{\text{loseknight}}(p) \wedge \underline{\text{location}}(p,bk_p,a) = 1$$
$$\wedge \underline{\text{location}}(p,kt_p,a) = 1$$
$$\wedge [(d_p(wk,bk)=3 \wedge d(bk_p,c(a))=3) \vee$$
$$(d_p(wk,bk)=4 \wedge d(bk_p,c(a))=4 \wedge$$
$$\underline{\text{location}}(p,wk_p,a)=-1)]\} .$$

There remains one more knight condition to define. This case occurs only in areas of size 4, and is illustrated by q_1 and p_2 in Figure 6.8. In p_2 the black king is able to escape from the area because the only move to block the escape, wk-K6, gives stalemate. p_2 is a successor to q_1. We recognize this pattern by

$$\underline{\text{bad4}}(p,a) \equiv \{\underline{\text{size}}(p,a)=4 \wedge \underline{\text{location}}(p,kt_p,a)=3$$
$$\wedge \underline{\text{location}}(p,bk_p,a)=1 \wedge d_p(bk,kt)=3\}$$
$$\underline{\text{bad4}}(q,a) \equiv \{\underline{\text{size}}(q,a)=4 \wedge \underline{\text{location}}(q,kt_q,a)=3$$
$$\wedge d(bk_q,c(a))=2\} .$$

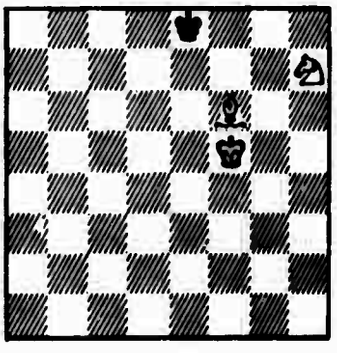
Now we can give a complete description of the conditions which an area must satisfy to be acceptable. We combine safe, badkt and bad4 into

q_1

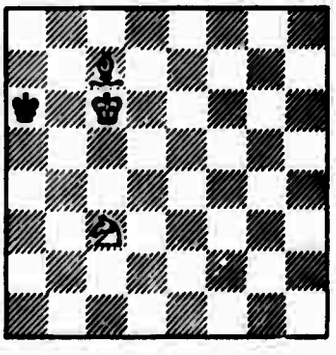


q_1 satisfies function bad4. If the black king moves to K1 (position p_2) white must permit him to escape from the area.

p_2



x_3



x_3 is an example of a stage 4 position.

Figure 6.8.

goodarea(x,a) ≡ (safe(x,a) ∧ ¬ badkt(x,a) ∧ ¬ bad4(x,a)) .

It is possible that more than one area in a position will satisfy goodarea. s will be the size of the smallest such area. Let C be a set containing the four corners of the board. Then we have

s(x) = min((size(x,a) | ∃ c(c∈C ∧ a=area(x,c) ∧ goodarea(x,a))) ∪ {15}) .

If no good area exists in x, s(x)=15 and x is in stage 2; otherwise s(x)<15 and x is in stage 3.

Stage 4

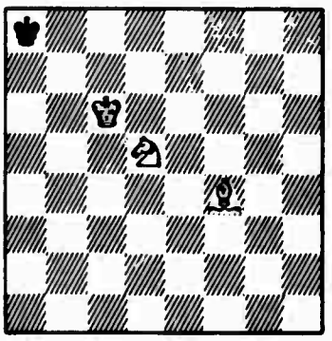
This stage is designed to be intermediate between stages 3 and 5. It is possible for the program to move into stage 5 (or even stage 6) directly. However, if black plays the best defense he will move toward the white corner and in that case the program will need stage 4 for at least two moves.

Position x_3 in Figure 6.8 is in stage 4. The black king is confined to the edge and completely controlled by the bishop and king. Function revcornpos(x) recognizes the pattern of these three pieces. Obviously revcornpos satisfies 3.7 and 3.9.

It is the position of the knight which determines that stage 4 rather than stage 5 holds. The bishop and king maintain control of the black king until the knight is in a position for stage 5.

Stage 5

Stage 5 controls the forcing of the black king down the edge of the board toward the corner where mate can be given. The play of the pieces in this stage must be very precise. The program follows closely the example from Capablanca [1935] given in Figure 6.9; it is interesting



The second and last part will consist in driving the Black King now from QR8 to QR1 or KR8 in order to mate him. QR1 will be the quickest in this position

10.	Kt–Kt6ch	K–R2
11.	B–B7	K–R3
12.	B–Kt8	K–R4
13.	Kt–Q5	K–R5

Black tries to make for KR1 with his King. White has two ways to prevent that, one by 14 B–K5, K–Kt6; 15 Kt–K3, and the other which I give as text, and which I consider better for the student to learn, because it is more methodical and more in accord with the spirit of all these endings, by using the King as much as possible.

14.	K–B5!	K–Kt6
15.	Kt–Kt4	K–B6
16.	B–B4	K–Kt6
17.	B–K5	K–R5
18.	K–B4	K–R4
19.	B–B7ch	K–R5
20.	Kt–Q3	K–R6
21.	B–Kt6	K–R5
22.	Kt–Kt2ch	K–R6
23.	K–B3	K–R7
24.	K–B2	K–R6
25.	B–B5ch	K–R7
26.	Kt–Q3	K–R8
27.	B–Kt4	K–R7
28.	Kt–B1ch	K–R8
29.	B–B3 mate	

It will be seen that the ending is rather laborious. There are two outstanding features: the close following by the King, and the controlling of the squares of opposite color to the Bishop by the combined action of the Knight and King. The student would do well to exercise himself methodically in this ending, as it gives a very good idea of the actual power of the pieces, and it requires foresight in order to accomplish the mate within the fifty moves which are granted by the rules.

Figure 6.9. Example from Capablanca, pages 110 and 111.

to note that this example is almost identical to the description of this part of the end game in all the other chessbooks we have examined.

During the play of this part of the game the white pieces must keep the black king close to the edge, and at the same time must force it toward the black corner. To simplify the pattern recognition, we limit the definitions, only recognizing enough positions to make the stage playable. Stage 5 will not contain all the positions occurring after white moves in Figure 6.9. As in stage 3, we will violate rule 3.9, but in this stage we can define a usable measure.

First of all we look for an edge e which satisfies the following predicates. Let $\underline{blc}(e)$ be the black corner on edge e . Then we have

6.3 $\underline{de}(bk_x,e)=0$.

6.4 $\underline{de}(wk_x,e)=2 \wedge (3 \leq \underline{dc}(wk_x,\underline{blc}(e)) \leq 7)$.

6.5 $\underline{de}(qb_x,e)\neq 2 \vee d(qb_x,\underline{blc}(e))>d(wk_x,\underline{blc}(e)))$.

Rule 6.3 says that the black king must be on the edge. Rule 6.4 says that the white king must be on the file/rank two away from the edge and also limits its position on that file or rank. For example if e is the QRfile, then the white king must be on the QBfile in one of the following squares: QB2, QB3, QB4, QB5, QB6. Rule 6.5 prevents the bishop from interfering with the movement of the white king down this file.

It is relatively easy to use the bishop and white king correctly in this game; the knight is a more difficult piece to control. For example the knight is the only piece which can be used to deny the black king a white square on the edge. If it is used to deny the black king a

black square on edge it probably will not be available for its correct use when it is needed. We adopt the following stringent condition:

6.6 $(\underline{\text{onblack}}(kt_x) \wedge \underline{de}(kt_x,e)=1) \vee (\neg\, \underline{\text{onblack}}(kt_x) \wedge \underline{de}(kt_x,e)=3)$.

Function $\underline{\text{onblack}}(sq)$ is true if the square sq is black. 6.6 allows the knight to bear only on white squares on the edge, and only on black squares on the file/rank next to the edge. One result of this is that we will be sure the white king is actually being used (once functions $\underline{conf}$ and $\underline{el}$ are defined) since it is the only piece which can bear on white squares in the file/rank next to the edge. Let function $\underline{eposs}(x,e)$ be true if rules 6.3 through 6.6 are satisfied.

In addition to rule 6.6, we also must be sure that the knight is close enough to the black king to be used effectively. First we must define a new distance function $\underline{fr}(sq1,sq2)$ which equals the difference in files between sq1 and sq2 plus the difference in ranks between sq1 and sq2 . For example in p_1 in Figure 6.10, $\underline{fr}_{p_1}(bk,kt)=4$. Then we have, for edge e

$$\underline{ktpos}(p,e) \equiv \{[\underline{\text{onblack}}(kt_p) \wedge \underline{fr}_p(bk,kt)\leq 2] \vee [\neg\, \underline{\text{onblack}}(kt_p) \wedge \underline{fr}_p(bk,kt)<5]\}$$

$$\underline{ktpos}(q,e) \equiv \{[\underline{\text{onblack}}(kt_q) \wedge d_q(wk,kt)\leq d_q(bk,kt) \wedge d(bk_q,\underline{blc}(e))-2\leq d(kt_q,\underline{blc}(e))\leq d(bk_q,\underline{blc}(e))+1] \vee [\neg\, \underline{\text{onblack}}(kt_q) \wedge \underline{fr}_q(bk,kt)=3]\} .$$

This condition, for the knight on a black square, prevents the knight from denying white edge squares to the black king from a position above the black king, because in that case the knight could not be used on the next move to keep the black king confined to the edge. The part

of ktpos which says $d_q(wk,kt) \leq d_q(bk,kt)$ prevents the bishop from being used when the white king should be.

So far we have defined the relationships between the white pieces, but we have not said exactly how they should control the black king. There are two parts to this control. First the black king must be prevented from escaping from the edge. A small escape may occur, as in black moves 14-16 in Figure 6.9, but we must be sure no larger escape is possible. For succ(x) the set containing the legal successors of the black king in x, we have

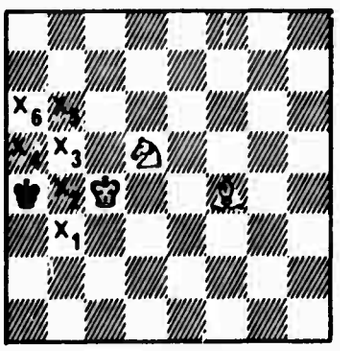
$$\underline{conf}(p,e) \equiv \forall r[r \in \underline{succ}(p) \supset (\underline{de}(r,e)=0 \vee \underline{fr}(bk_p,r)=2)]$$

$$\underline{conf}(q,e) \equiv \forall r[r \in \underline{succ}(q) \supset \underline{de}(r,e)=0] .$$

conf is only concerned with the squares labeled X1, X2 and X3 in p_1. in Figure 6.10. In q both squares are denied to the black king; in p only X2 is denied.

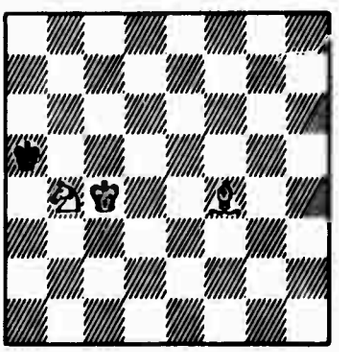
The control of squares X3, X4, X5 and X6 is measured by function el. The function determines the amount of control the white pieces have on the black king from above. To define el, we need function bears(x,X) which is true if the white pieces in x bear on square X, or if X is not on the board. In the following definition XN stands for a function with arguments (position,edge,N) which produces the appropriate square, or NIL if the square is off the board. onblack(NIL)=NIL. The following definition assumes that conf is satisfied. We have

p_1



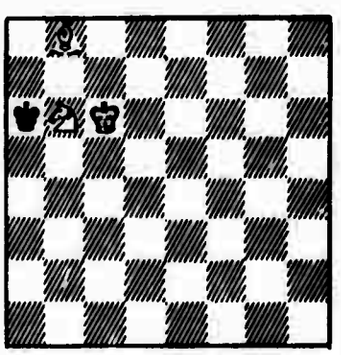
el(p_1,QRfile) = 2 and dedge(p_1) = 5.
kt—QKt4 will give a q with dedge(q) = 4.

p_2



el(p_2,QRfile) = 0 and dedge(p_2) = 4.
qb—QB7 will give a q with dedge(q) = 3.

p_3



Positions p_3 and p_4 are not accepted by stage 5.

p_4

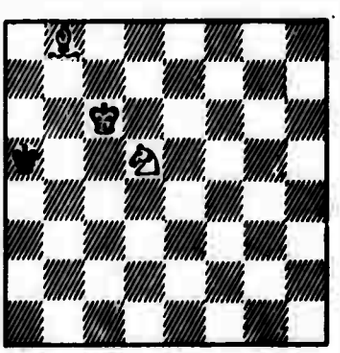


Figure 6.10. Examples for Stage 5.

$\underline{el}(p,e) = 0$ if $[\underline{bears}(p,X4) \wedge (\underline{onblack}(X3) \vee \underline{bears}(p,X3))]$.

$= 1$ if $[\neg \underline{bears}(p,X4) \wedge \underline{bears}(p,X3)$

$\wedge \underline{bears}(p,X6) \wedge (\underline{bears}(p,X5) \vee \underline{onblack}(X5))]$.

$= 2$ if $[\neg \underline{bears}(p,X4) \wedge \underline{bears}(p,X3) \wedge \neg \underline{bears}(p,X6)$

$\wedge (\underline{bears}(p,X5) \vee \underline{onblack}(X5))]$.

$= 3$ otherwise.

$\underline{el}(q,e) = 0$ if $[\underline{bears}(q,X4)]$.

$= 1$ if $[\neg \underline{bears}(q,X4) \wedge \underline{bears}(q,X6)$

$\wedge (\underline{onblack}(X5) \vee \underline{bears}(q,X5))]$.

$= 2$ if $[\underline{bears}(q,X5) \wedge \neg \underline{bears}(q,X4)$

$\wedge \neg \underline{bears}(q,X6)]$.

$= 3$ otherwise.

$\underline{el}(x,e)<3$ means sufficient control from above exists in x . Combining this with conf, we have in q that the black king must be confined to the edge. In p it cannot escape the edge into X2 , the square next to its present position; rule 3.9, may be violated at this point. If it can escape above, the escape square must be black. This is necessary to accomodate a position like the one in Figure 6.9 after black move 18. A white square off the edge is not permitted to the black king, even in two moves. Only the white king can control such a square. Values of el are given in Figure 6.10 for positions p_1 and p_2 .

Finally there are two positions p which satisfy all the conditions given so far, but cannot be handled by the ordinary rules. They are illustrated by p_3 and p_4 in Figure 6.10. The problem is one of parity; if identical positions to p_3 and p_4 occurred farther down on the same edge, the bishop would be able to make a move while

continuing to bear on the same edge square. We recognize p_3 and p_4 by $\underline{badedge}(p,e)$, and $\underline{badedge}(q,e) \equiv$ false for all q.

Now we can give a complete definition of a good edge. Let E be a set containing the four edges of the board. Then we have

$\underline{edge}(x) = e$ if $[e \in E \wedge \underline{eposs}(x,e) \wedge \underline{ktpos}(x,e) \wedge \underline{conf}(x,e) \wedge \underline{el}(x,e) < 3 \wedge \neg \underline{badedge}(x,e)]$.

$=$ NIL if no such e exists.

A position x is in stage 5 if $\underline{edge}(x)$ is not null.

Next we define a measure for stage 5. This is an indicator of how much access the black king has to the white corner. For p we can use $d(bk_p, \underline{blc}(\underline{edge}(p))) + \underline{el}(p, \underline{edge}(p))$. For q we must make some adjustments in this formula. We define

$\underline{adj}(q,e) = -1$ if $[\underline{el}(q,e)=0 \wedge$ (the black king is in check in q)].

$= +1$ if $[\underline{el}(q,e)=2 \wedge \underline{onblack}(bk_q) \wedge \neg \underline{onblack}(kt_q)]$.

$= 0$ otherwise.

$\underline{adj}(p,e) = 0$

Then we have

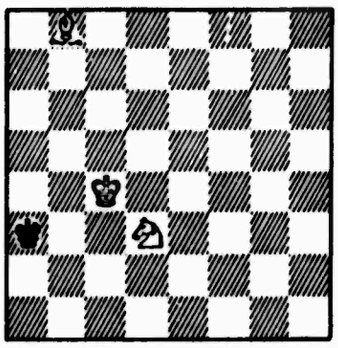
$\underline{dedge}(x) = d(bk_x, \underline{edge}(x)) + \underline{el}(x, \underline{edge}(x)) + \underline{adj}(x, \underline{edge}(x))$

and $\underline{dedge}$ is a measure for stage 5. For example in p_1 in Figure 6.10, $\underline{edge}(p_1) =$ QRfile and $\underline{dedge}(p)=5$. Only kt-QKt4 will give a q in stage 5, and $\underline{dedge}(q)=4$. Therefore this q will be accepted by $\underline{better}$. For either black king move in this q, we will get a p with $\underline{dedge}(p)=4$. p_2 in Figure 6.10 is one of these successors. In p_2, wk-QB5 and qb-K3 give positions in stage 5 with $\underline{dedge}=4$; however qb-QB7 will give $\underline{dedge}=3$.

It is not difficult to show that edge and dedge satisfy rule 3.7. For edge, the only condition which presents any difficulty is el and the value of el determines dedge. We must consider cases. If el(q,e)=0 , then the black king must move toward the black corner, giving el(p,e)=0 or 1 , depending on whether the black king was in check in q . In either case dedge(p) = dedge(q) . If el(q,e)=1 and the black king moves down the edge then there is no problem and el(p,e)=2 ; if the black king moves away from the black corner, we have a position like p_2 in Figure 6.10, with el(p,e)=0 . There is no danger that a white square off the edge and above is available to the black king in p , because this is expressly forbidden in q . Again we have dedge(p)=dedge(q) . If el(q,e)=2 , we must have a position like q_1 or q_2 in Figure 6.11. The black king can move down only in q_1 , and we will obviously get el(p,e)=2 and dedge(p)<dedge(q) ; if the black king moves up in q we will get el(p,e)=1 or 2 depending on adj(q) . In q_1 , with $adj(q_1)=1$, we get el(p,e)=2 , while in q_2 , with $adj(q_2)=0$, we get el(p,e)=1 . In either case dedge(q)=dedge(p) . Therefore all of rule 3.7 is satisfied.

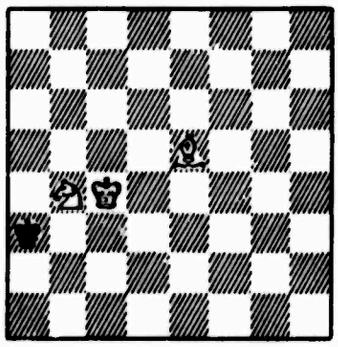
We cannot hope to satisfy rule 3.9 because sometimes a p will have all successors in a lower stage. For example this occurs after black move 13 in Figure 6.9. As explained before, this is not critical to the working of the program. The reason we can use dedge as a measure in this stage is because there is no error in the evaluation of dedge, and if dedge(q)>dedge(p) , there really has been a loss of control.

q_1



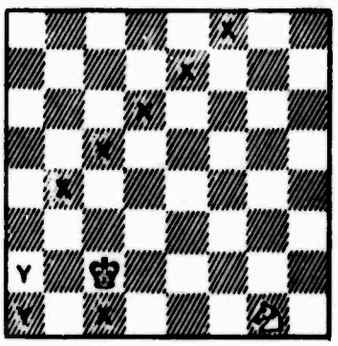
$\underline{el}(q_1, QRfile) = 2$ and $\underline{dedge}(q_1) = 5$.

q_2



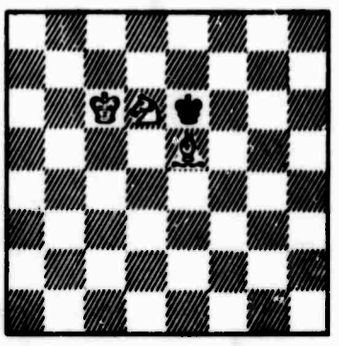
$\underline{el}(q_2, QRfile) = 2$ and $\underline{dedge}(q_2) = 4$.

x_3



For stage 6, the bishop must be in a square marked X , and the black king in a square marked Y; the white king must be as shown. The position of the knight is not important.

p_4



kt–QB4 preserves the area and protects the bishop. However, $d_q(wk,qb) > d_q(bk,qb)$, so q is in stage 1.

Figure 6.11.

Stage 6

Stage 6 is similar to stage 4 in that the white king and bishop control the black king, while the knight is maneuvered into position for the next stage (checkmate). Position x_3 in Figure 6.11 is an example of a stage 6 position. The relative positions of the white king and bishop and the black king are recognized by cornerpos(x). Obviously, cornerpos satisfies 3.7 and 3.9.

Formal Definitions of better and worse

Now we can give the definitions of stages and measures. The stages are

$x \in$ stage 0 $\equiv$ {(x is a position with black to move) $\wedge$
[(x is stalemate) $\vee$ (the black king can take a piece in one move in x) $\vee$ badpos(x)]}.

$x \in$ stage 1 $\equiv$ stage1 (x).

$x \in$ stage 2 $\equiv$ {$\neg$ stage1(x) $\wedge$ s(x) = 15}.

$x \in$ stage 3 $\equiv$ {$\neg$ stage1(x) $\wedge$ s(x) < 15}.

$x \in$ stage 4 $\equiv$ revcornpos(x).

$x \in$ stage 5 $\equiv$ edge(x)

$x \in$ stage 6 $\equiv$ cornerpos(x)

$x \in$ stage 7 $\equiv$ x is checkmate.

The measures are

$m_2(x)$ = dcent(x) $\quad$ $x \in$ stage 2.

$m_5(x)$ = dedge(x) $\quad$ $x \in$ stage 5.

$m_i(x) = 0$ $\quad$ $x \in$ stage i, i = 0, 1, 3, 4, 6, 7

Additions to better and worse

The formal definition of better is grossly inadequate only in stage 3. In the other stages additions may be needed in worse. No changes will be made in stage 1 since it is very short.

In stage 2 when dcent(p)>0 we ordinarily expect a tree search of no more than depth 2. If the tree search is longer, this will mean we are moving the knight out of danger and so the tree will be quite narrow. When dcent(p)=0 the tree may be deeper, since several moves may be required to establish s(q)<15 . We can eliminate many bishop moves by

6.7 $\underline{st}(p)=2 \wedge \underline{st}(q)\leq 2 \wedge qb_p \neq qb_q \wedge d_q(wk,qb)>2 \wedge (\underline{dcent}(p)=0 \vee \underline{st}(q)=2)$.

6.7 is defined for all values of dcent(p) because when dcent(p)>0 , we are not interested in bishop moves except to protect the bishop. There will always be time to make these protective moves without violating 6.7 because if there were not, we would be in stage 1.

Stage 3 may require more than 20 moves. We immediately add to better

6.8 $\underline{st}(p)=\underline{st}(q)=3 \wedge s(q)<s(p)$

because as previously noted the difficulty with s as a measure involves worse (it violates rule 3.9 but satisfies rule 3.7 which is the critical one for better). However even with 6.8, more than ten moves may be needed to force a smaller area. Both the length and the breadth of the tree search must be decreased. In the following discussion ar(s(x)) gives the area for which s is the size.

We can eliminate many moves by adding to worse

6.9 $\{\underline{st}(p)=3 \wedge \underline{st}(q)<3$

$\wedge (d_q(wk,qb)>2 \vee [\underline{st}(q)=1 \wedge \neg \underline{ktspec}(p,q)])\}$

where

$\underline{ktspec}(p,q) \equiv \{d_p(kt,qb)=1 \wedge \underline{location}(p,kt_p,\underline{ar}(s(p)))=0$

$\wedge \underline{losebishop}(p) \wedge d_p(wk,kt)=1 \wedge \underline{fr}_p(wk,qb)=3$

$\wedge wk_p=wk_q \wedge qb_p=qb_q \wedge \neg \underline{onblack}(kt_q)\}$.

ktspec recognizes a position like p_4 in Figure 6.11. All moves but kt-QB4 will be rejected at depth 1 since either we would have a q in stage 0, or $d_q(wk,qb)>2$. The last three requirements of ktspec eliminate moves farther down in the tree.

6.9 does not provide sufficient pruning to permit the program to handle a tree of depth 10. We can shorten the tree by considering how the program must move to force a smaller area. It does this by coordinating the action of the three pieces. We recognize certain of the patterns involved by means of function v defined for x in stage 3. We have

$v(x) = 1$ if $\neg \underline{ktv1}(x)$

$= 3$ if $\underline{ktv1}(x) \wedge \neg \underline{ktv2}(x)$

$= 5$ if $\underline{ktv1}(x) \wedge \underline{ktv2}(x)$

where

$\underline{ktv1}(x) \equiv \{\underline{location}(x,kt_x,\underline{ar}(s(x)))=-2$

$\wedge d(kt_x,c(\underline{ar}(s(x))))=s(x)\}$

$\underline{ktv2}(x) \equiv \{[s(x)=4 \wedge d_x(wk,kt)>1]$

$\vee [s(x)>4 \wedge d_x(qb,kt)=3 \wedge d_x(wk,kt)=3$

$\wedge d_x(wk,qb)=1]\}$.

Figure 6.12 gives examples of $v=3$ and $v=5$, for $s=5$. We can use v by adding to better

6.10 $\underline{st}(p)=\underline{st}(q)=3 \wedge s(q)=s(p) \wedge v(q)>v(p)$.

6.10 cuts down the depth of tree search in almost all cases to a maximum of 6. This maximum is exceeded when the black king is able to escape from the area in p. This escape will either result in a smaller area, or will quickly be blocked. In the latter case, the moves used to block the escape must be added to the moves required to increase the value of v. Position p_3 in Figure 6.12 is an example; a tree of depth 8 is required. We can reduce this as follows. We define function $\underline{poss}(p)$ which is true if the black king can escape in one move from p. Then

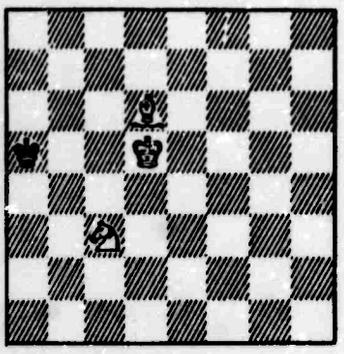
6.11 $\underline{st}(p)=\underline{st}(q)=3 \wedge s(p)=s(q) \wedge \underline{poss}(p) \wedge \underline{ktposs}(p,q)$

can be added to better. Function ktposs handles a position like p_4 in Figure 6.12. If the program simply accepted any q in stage 3 with the same size area, then it would accept one with the knight still on the boundary of the area, so the whole tree would have to be repeated. ktposs will reject such a q.

The addition of 6.11 to better insures a maximum depth of 6 for trees in stage 3. Considerable pruning will be needed before the program can handle these trees. As an aid to pruning we introduce function sl for positions q in stage 2. $\underline{sl}(q)$ is the size of the smallest area a in which

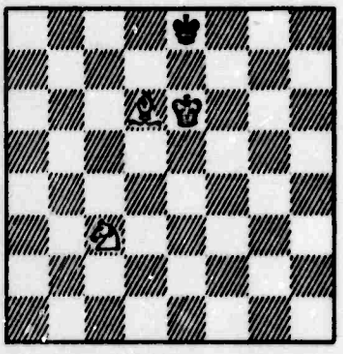
6.12 $\underline{inside}(q,a) \wedge \neg\, \underline{badkt}(q,a) \wedge \neg\, \underline{bad4}(q,a)$

holds. $\underline{sl}(q)=15$ if no area in q satisfies 6.12. If $s(q)=15$ and



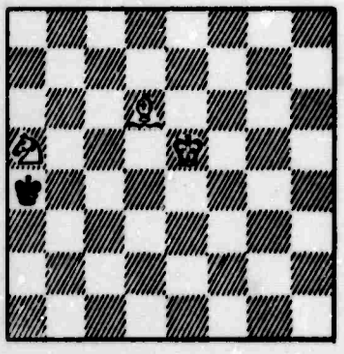
p_1

$s(p_1) = 5$, $v(p_1) = 3$. The knight is guarding part of the boundary of the area, which frees the king so it can force a smaller area.



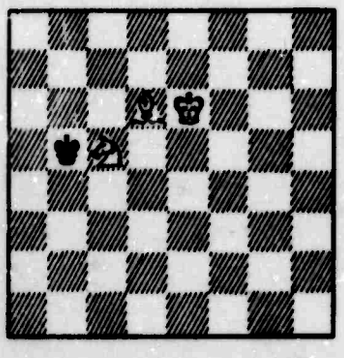
p_2

$s(p_2) = 5$, $v(p_2) = 5$. The king must do the forcing on the part of the boundary away from the knight.



p_3

$s(p_3) = 5$ and $v(p_3) = 1$. White must move the knight (to QB4) and the black king can escape to QKt6. poss(p_3) is true. If the king were in K6, it would be unable to later block the escape and the position would be in stage 2.



p_4

$s(p_4) = 5$ and $v(p_4) = 1$ and poss(p_4). If after wk–Q5 and bk–QKt3, then wk–K5, the resulting q will be rejected by ktposs.

Figure 6.12.

$\underline{sl}(q) \leq 6$, this means either $\underline{bpos}$ or $\underline{kpos}$ failed for the area. One possibility is that $\underline{sl}(q)=3$ ($\underline{bpos}$ cannot be satisfied in this case).

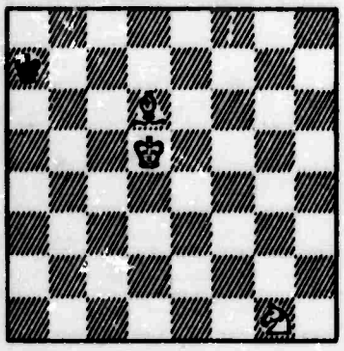
When $\underline{st}(q)=2$, we look at $\underline{sl}(q)$. If $\underline{sl}(q)<s(p)$ either white is blocking an escape by forcing a smaller area, in which case $\underline{sl}(q)=s(p)-1$, or white is trying to make a smaller area by moving the bishop toward the corner, giving $\underline{sl}(q)=s(p)-2$. Often such a move is wasted because the black king will easily escape. When $s(p)=4$ we eliminate both kinds of moves; in addition we reject the second kind when $\underline{sl}(q)=3$ unless $v(p)=5$ (in this case it is an interim move to stage 4). We also eliminate positions with unlikely king locations. We reject all positions satisfying

$$\underline{badsmall}(p,q) \equiv \{s(p)=4 \vee [s(p)=\underline{sl}(q)+2 \wedge$$
$$([\underline{sl}(q)=3 \wedge v(p)<5] \vee \underline{location}(p, wk_q, \underline{ar}(s(p))) \neq 1$$
$$\vee d(wk_q, c(\underline{ar}(s(p)))) > 3)]\} .$$

We divide the remainder of the discussion of $\underline{st}(p)=3$ into two parts: $v(p)=1$ and $v(p)>1$. For $v(p)>1$, we can be very concise in our descrfption of bad moves. When $v(p)=3$ we refuse all moves such that

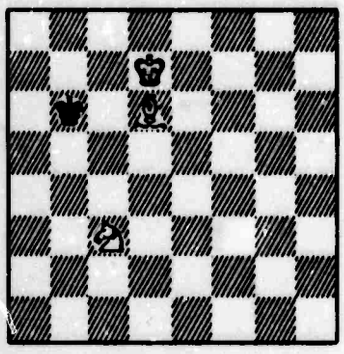
6.13 $(s(q)=s(p) \wedge v(q)=1) \vee (\underline{st}(q)=2 \wedge \underline{sl}(q)>s(p))$.

We permit $\underline{st}(q)=2$ only if $\underline{sl}(q)=s(p)$. This occurs when the white king has moved into the area to try to force a smaller area without the aid of the knight. Position p_1 in Figure 6.13 is an example of a place where such a move should be made. Again this kind of move will often be wasted since the black king can easily escape. We reject all such q satisfying



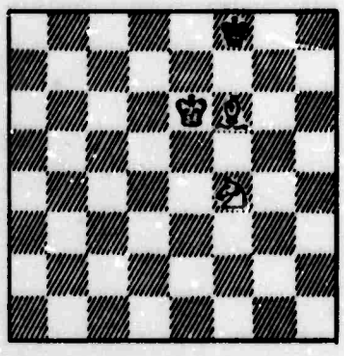
p_1

wk–QB6 is the best move, and on the next move, qb–QB7 will give stage 4.



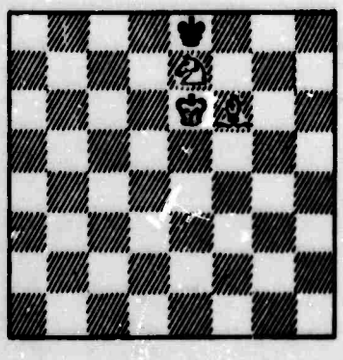
p_2

When v(p) = 5, it is time to move the white king inside the area to the square indicated in p_2. $s(p_2) = 15$, but $\underline{s1}(p_2) = 5$.



p_3

$s(p_3) = 4$ and $v(p_3) = 5$. The knight moves to Q5; on the next move, it may go to K7 giving q_4.



q_4

$\underline{st}(q_4) = 4$ and $\underline{s1}(q_4) = 15$. We are almost in stage 5, but need to move the bishop.

Figure 6.13.

$$\underline{\text{trysmall}}(p,q) \equiv \{qb_p \neq qb_q \vee \underline{\text{location}}(p, wk_q, \underline{\text{ar}}(s(p))) \neq 1$$
$$\vee [d_q(wk,bk)=s(p)-3 \wedge (d_q(wk,\acute{q}b)>1$$
$$\vee \underline{\text{location}}(p,bk_q,\underline{\text{ar}}(s(p)))<s(p)-1)]$$
$$\vee d_q(wk,bk)>s(p)-4 \vee [d_q(wk,bk)=s(p)-4$$
$$\wedge \underline{\text{location}}(p,bk_q,\underline{\text{ar}}(s(p)))<s(p)-2]\} .$$

For example, if in p_1 the black king were in QR3, the white move wk-QB6 would be rejected by trysmall.

When $v(p)=5$, the tree is fairly long, up to depth 6. First, we introduce a rule similar to 6.13. q will be worse than p if

6.14 $(s(q)=s(p) \wedge v(q)=3) \vee (\underline{st}(q)=2 \wedge \underline{sl}(q)>s(p))$.

We can decide what other moves to reject by considering how the program should play. We want to move the white king inside the area to form a position like p_2 in Figure 6.13. The knight is protecting the boundary of the area, so we need not worry that the black king will escape when we do this. Sometimes it will be necessary to move the knight before the king move can be made. This knight move is a tempo move; it must satisfy

$$\underline{\text{ktmove}}(p,q) \equiv \{kt_p=kt_q \vee [d_q(wk,bk)=2 \wedge$$
$$(s(p)>4 \wedge \underline{\text{location}}(p,kt_q,\underline{\text{ar}}(s(p)))=-1$$
$$\wedge d(kt_q,c(\underline{\text{ar}}(s(p))))=s(p)-2) \vee$$
$$(s(p)=4 \wedge \underline{\text{location}}(p,kt_q,\underline{\text{ar}}(s(p)))=0$$
$$\vee [\underline{\text{location}}(p,kt_q,\underline{\text{ar}}(s(p)))=-3 \wedge$$
$$d(kt_q,c(\underline{\text{ar}}(s(p))))=4])]\} .$$

When $s(p)>4$, only one knight move is permitted. In p_2 this is the move kt-Q5. When $s(p)=4$ an additional knight move must be allowed owing to the peculiarities of stage 5. Positions p_3 and q_4 in

Figure 6.13 are examples. One result of the second knight move is that 6.14 must be amended so that a position like q_4 will not be rejected ($\underline{sl}(q_4)>4$). Instead of 6.14 we have

$$\{(s(q)=s(p) \wedge v(q)=3) \vee (\underline{st}(q)=2 \wedge \underline{sl}(q)>s(p)$$
$$\wedge [s(p)>4 \vee kt_p=kt_q \vee \neg \underline{ktmove}(p,q)])\} .$$

In addition to the knight, the bishop makes a tempo move. p_2 in Figure 6.13 is an example. However we can limit the number of bishop moves allowed by refusing those satisfying

$$\{qb_p \neq qb_q \wedge [d(qb_q,qb_p)>1 \vee d_q(wk,bk)>2 \vee \underline{bpos}(q,\underline{ar}(s(p)))]\} .$$

Finally we can reject many king moves (and an occasional bishop move) by

$$\underline{badkmove}(p,q,lo) \equiv \{lo<-1 \vee lo>1 \vee [lo<1 \wedge (d_q(wk,bk)>2$$
$$\vee qb_p \neq qb_q \vee [lo=-1 \wedge wk_p \neq wk_q])]\} ,$$

where

$$lo=\underline{location}(p,wk_q,\underline{ar}(s(p))) .$$

We combine all these conditions for $v(p)>1$, excepting $s(q)<s(p)$ or $\underline{sl}(q)<s(p)$ in

$$\underline{check3b}(p,q) \equiv \{(\underline{st}(q)=2 \wedge \underline{sl}(q)>s(p) \wedge [s(p)>4 \vee v(p)<5$$
$$\vee kt_p=kt_q \vee \neg \underline{ktmove}(p,q)])$$
$$\vee (v(p)=3 \wedge [\underline{st}(q)=3 \wedge v(q)=1]$$
$$\vee [\underline{st}(q)=2 \wedge \underline{sl}(q)=s(p) \wedge \underline{trymove}(p,q)])$$
$$\vee (v(p)=5 \wedge [\underline{st}(q)=3 \vee \underline{sl}(q)=s(p)]$$
$$\wedge [\neg \underline{ktmove}(p,q) \vee (\underline{st}(q)=3 \wedge v(q)=3) \vee$$
$$\underline{badkmove}(p,q,\underline{location}(p,wk_q,\underline{ar}(s(p))))])\} .$$

When $v(p)=1$ white does not have much control. All knight moves must be permitted except those giving stage 0 or stage 1. When

$s(q)=s(p)$, we limit the number of moves somewhat by

$$s(q)=s(p) \wedge d_q(wk,bk)>d(wk_p,bk_q) .$$

$\underline{st}(q)=2$ or $s(q)>s(p)$ is only permissible when the black king is able to escape from the area in p . Position p_1 in Figure 6.14 is an example. Then q is an intermediate position on a branch of the tree leading either to a smaller area or the same area under better control. We can limit moves by

$$\{\neg \underline{poss}(p) \vee \neg \underline{kpos}(q,\underline{ar}(s(p))) \vee d_q(wk,bk)>d_p(wk,bk) \vee \underline{location}(p,wk_q,\underline{ar}(s(p)))>-1\} .$$

We combine conditions for $v(p)=1$ excepting $s(q)<s(p)$ or $\underline{sl}(q)<s(p)$ in

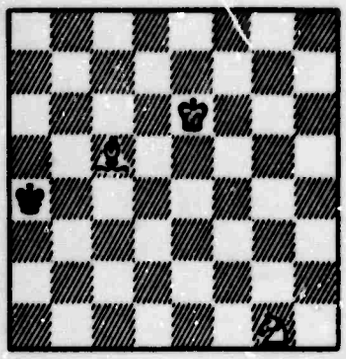
$$\begin{aligned}\underline{check3a}(p,q) \equiv \{&(\underline{st}(q)=3 \wedge s(q)=s(p) \wedge d_q(wk,bk)>d_p(wk,bk))\\ &\vee (\underline{st}(q)=2 \wedge \underline{sl}(q)=s(p) \wedge \underline{trysmall}(p,q))\\ &\vee ([(\underline{st}(q)=3 \wedge s(q)>s(p)) \vee (\underline{st}(q)=2 \wedge \underline{sl}(q)>s(p))]\\ &\quad \wedge [s(p)=4 \vee qb_p \neq qb_q \vee \neg \underline{poss}(p)\\ &\qquad \vee \neg \underline{kpos}(q,\underline{ar}(s(p))) \vee d_q(wk,bk)>d_p(wk,bk)\\ &\qquad \vee \underline{location}(p,wk_q,\underline{ar}(s(p)))>-1])\} .\end{aligned}$$

The heuristics for p in stage 3 in worse are

$$\begin{aligned}\underline{check3}(p,q) \equiv \{&\underline{st}(q)\leq 3 \wedge [d_q(wk,qb)>2 \vee\\ &(\underline{st}(q)=1 \wedge \neg \underline{ktspec}(p,q)) \vee\\ &(\underline{st}(q)=2 \wedge \underline{sl}(q)<s(p) \wedge \underline{badsmall}(p,q)) \vee\\ &([(\underline{st}(q)=3 \wedge s(q)\geq s(p)) \vee (\underline{st}(q)=2 \wedge \underline{sl}(q)\geq s(p))]\\ &\wedge [(v(p)>3 \wedge \underline{check3b}(p,q)) \vee\\ &\quad (v(p)=1 \wedge \underline{check3a}(p,q))])]\} .\end{aligned}$$

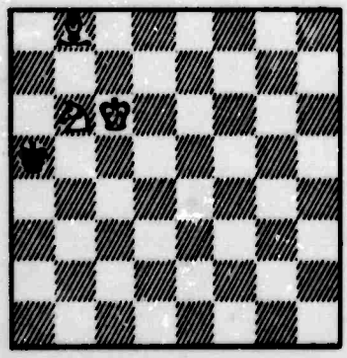
In better we add 6.8, 6.10 and 6.11.

p_1



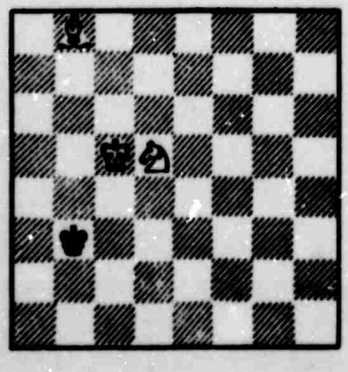
$s(p_1) = 5$, but when wk–Q5 we will have $s(q) = 6$. We permit this move. $d_q(wk, bk) < d_{p_1}(wk, bk)$ and $\underline{poss}(p_1)$.

p_2



This position is at the head of the major tree search in stage 5(depth 7). kt–Q5 is the only move on the first level.

p_3



This position occurs down in the tree from p_2, after 2 white and 2 black moves. kt–QKt4 is the only move permitted.

Figure 6.14.

In stage 4 ordinarily a tree of depth 2 will be required to reach stage 5 because we expect to enter stage 4 from stage 3 with the knight appropriately placed for $s=5$. However we may occasionally have stage 4 in a starting position or enter it from $s=5$ before the knight is put in position. In such a case a tree search of up to depth 5 may occur. Since the whole point of stage 4 is that the white king and bishop can control the black king without moving, allowing white to bring the knight into play, we can easily reduce the breadth of the tree search by adding to worse:

$$\underline{st}(p)=4 \wedge \underline{st}(q)<4$$

Then trees in stage 4 will be almost all knight moves.

In stage 5 tree searches are very short except for the one black attempt to escape from the edge (moves 14-16 in Figure 6.9), when a tree of depth 7 is required. p_2 in Figure 6.14 is the position at the head of the tree. We first of all eliminate all positions q with $\underline{st}(q)\neq 2$. Before proceeding further we must be able to recognize the edge e even in positions where the black king is not on an edge. We look for q in stage 2 such that, for $e=\underline{edge}(p)$

$$\begin{aligned}\underline{kcond}(q,e) \equiv \{&\underline{eposs}(q,e) \wedge [\underline{de}(bk_q,e)=0 \wedge \underline{el}(q,e)<3 \\ &\wedge d(bk_q,\underline{bk}(e))\leq 3 \wedge \underline{kconda}(q,e)] \\ &\vee [\underline{de}_q(bk,e)>0 \wedge d(bk_q,\underline{blc}(e))\leq 2 \wedge \underline{kcondb}(q,e)]\}\end{aligned}$$

where

$$\underline{kconda}(q,e) \equiv \underline{fr}_q(bk,kt)<5 \wedge (\underline{onblack}(bk_q) \vee \underline{fr}_q(wk,bk)<4)$$

and

$$\begin{aligned}\underline{kcondb}(q,e) \equiv \{&d_q(wk,bk)\leq 3 \wedge([\underline{de}(bk_q,e)=1 \wedge \\ &\forall r(r\in\underline{succ}(q) \supset [\underline{de}(r,e)<2 \vee \underline{fr}(wk_q,r)=2])] \\ &\vee [\underline{de}(bk_q,e)=2 \wedge \forall r(r\in\underline{succ}(q) \supset \underline{de}(r,e)=1)])\} .\end{aligned}$$

These conditions insure that the white pieces remain in the proper locations for stage 5. In addition, they are so stringent that they often prevent the many bishop moves (the bishop is the least constrained piece in stage 5) simply because one of the other pieces has to move. Positions p_2 and p_3 in Figure 6.14 are examples. In p_2, only kt-Q5 will be permitted and in p_3 only kt-QKt4. In fact the effect of these rules is to reduce the tree to almost one branch. Occasionally a few bishop moves will be considered but they are down in the tree where they do not do much harm. Since the tree has only one branch we could decide on many of the moves without tree search. However handling them through tree search enables the program to avoid extra pattern recognition of the positions with white to move which would result from such positions. Summing up these rules, we add to worse

$$\{\underline{st}(p)=5 \wedge \underline{st}(q)<5 \wedge (\underline{st}(q)\neq 2 \vee \neg\, \underline{kcond}(q, \underline{edge}(p)))\} .$$

Stage 6 is similar to stage 4, and we immediately add to worse

$$\underline{st}(p)=6 \wedge \underline{st}(q)<6 .$$

However this may permit four bishop moves at every level in addition to all the knight moves, and although usually the tree is only of depth 3 or 4, it may be longer. We must allow one bishop move for parity, but we eliminate all others by insisting that they satisfy

$$\underline{bcorner}(p,q) \equiv \{d_q(qb,wk)<3 \vee (d_p(qb,wk)=2 \wedge d_q(qb,wk)=3)\} .$$

Combining the formal definitions of better and worse with the various additions we have

$$\underline{better}(p,q) \equiv [\underline{st}(p)<\underline{st}(q) \vee [\underline{st}(p)=\underline{st}(q) \wedge m_{\underline{st}(p)}(q)<m_{\underline{st}(p)}(p)]$$
$$\vee [\underline{st}(p)=3 \wedge (s(q)<s(p) \vee$$
$$[s(q)=s(p) \wedge (v(q)>v(p) \vee \underline{poss}(p))])]]$$

$$\underline{worse}(p,q) \equiv [\underline{st}(q)=0 \vee (\underline{st}(p)=\underline{st}(q) \wedge m_{\underline{st}(p)}(q)>m_{\underline{st}(p)}(p))$$
$$[\underline{st}(q)\leq \underline{st}(p) \wedge$$
$$[(\underline{st}(p)=2 \wedge qb \neq qb_p \wedge d_q(wk,qb)>2 \wedge [\underline{dcent}(p)=0 \vee \underline{st}(q)=2])$$
$$\vee (\underline{st}(p)=3 \wedge [d_q(wk,qb)>2 \vee (\underline{st}(q)=1 \wedge \neg \underline{ktspec}(p,q))$$
$$\vee (\underline{st}(q)>1 \wedge \underline{check3}(p,q))])$$
$$\vee (\underline{st}(p)=4 \wedge \underline{st}(q)<4)$$
$$\vee (\underline{st}(p)=5 \wedge \underline{st}(q)<5 \wedge$$
$$[\underline{st}(q)\neq 2 \vee \neg \underline{kcond}(q,\underline{edge}(p))])$$
$$\vee (\underline{st}(p)=6 \wedge [\underline{st}(q)<6 \vee \neg \underline{bcorner}(p,q)])]].$$

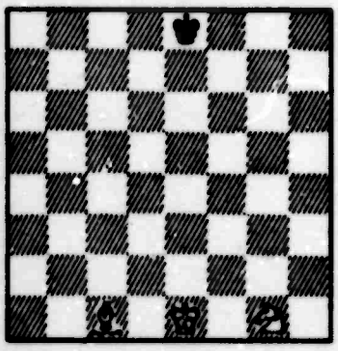
These functions are equivalent to the definitions used by the program.

Examples of Program Play

Our first example starts with position p_1 in Figure 6.15. $\underline{st}(p_1)=2$ and $m_2(p_1)=3$.

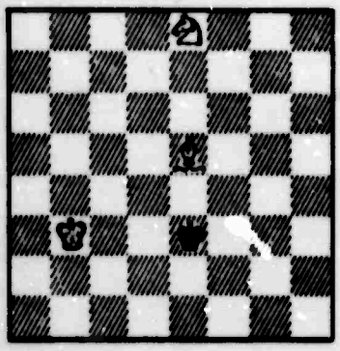
1.	wk-Q2	bk-K2	
2.	wk-QB3	bk-K3	
3.	wk-Q4		We now have $m_2=0$.
		bk-Q3	
4.	qb-KB4 ch.	bk-QB3	
5.	qb-K5	bk-Q2	
6.	wk-Q5		Now we are in stage 3 with an area of size 6. Moves 4, 5 and 6 are selected by tree search.
		bk-K2	

p_1



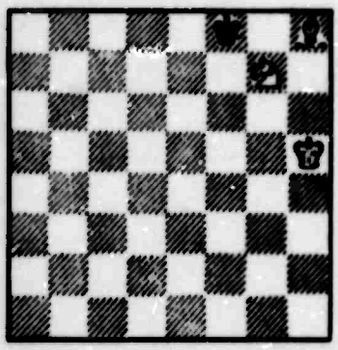
p_1 is in stage 2, and dcent(p_1) = 3.

p_2



p_2 is in stage 2 and dcent(p_2) = 2.

p_3



p_3 is in stage 2 and dcent(p_3) = 3. However all immediate successors of p_3 are in stage 0 or 1.

Figure 6.15. Starting Positions for Example of Program Play.

7.	kt-KR3	bk-KB2	
8.	kt-KB2		Now v=5 . We have skipped over v=3 .
		bk-K1	
9.	wk-K6	bk-Q1	
10.	qb-Q6		Moves 9 and 10 are selected by a tree search of depth 2. Now we have an area of size 5.
		bk-QB1	
11.	kt-KKt4	bk-QKt2	
12.	kt-KB6		Now v=3 .
		bk-QKt3	
13.	wk-Q5	bk-QR3	
14.	wk-QB4	bk-QKt3	
15.	qb-QB5 ch.		Now v=5 . Moves 13, 14 and 15 are selected by a tree of depth 3.
		bk-QKt2	
16.	wk-QKt5	bk-QB2	
17.	qb-QKt4		This is the bishop move allowed for tempo.
		bk-QKt2	
18.	qb-QR5		Now we have sl=3 .
		bk-QR2	
19.	wk-QB6	bk-QR3	
20.	qb-QR7		Now we are in stage 4. Moves 16 through 20 are selected by a tree of depth 4.
		bk-QR2	
21.	kt-Q7	bk-QR1	

22.	kt-QKt6 ch.		We are in the same position as Figure 6.9 after white move 10.
		bk-QR2	
23.	qb-Q6		A tempo move.
		bk-QR3	
24.	qb-QKt8		$m_5=5$.
		bk-QR4	
25.	kt-Q5	bk-QR5	
26.	wk-QB5	bk-QR6	
27.	kt-QKt4	bk-QKt7	
28.	qb-KB4	bk-QKt8	
29.	wk-QB4	bk-QKt7	
30.	qb-K3		This is the only place in the tree starting at move 25 where more than one white move is considered.
		bk-QR6	
31.	qb-Q4		Now we have reached the end of the branch of the tree (of depth 7) and $m_5=4$.
		bk-QR5	
32.	qb-QKt6		$m_5=3$.
		bk-QR6	
33.	kt-Q3	bk-QR5	
34.	kt-QKt2 ch.		$m_5=2$.
		bk-QR6	
35.	wk-QB3	bk-QR7	
36.	wk-QB2	bk-QR6	

37.	qb-QB5 ch.		m_5=1 . Moves 35, 36 and 37 are selected by a tree of depth 3.
		bk-QR7	This move gives a p in stage 6.
38.	kt-Q3	bk-QR8	
39.	qb-QKt4		A tempo move.
		bk-QR7	
39.	kt-QB1 ch.	bk-QR8	
40.	qb-QB3 mate.		

The program plays the last part of the game (from move 22 on) identically to Figure 6.9; different black moves have been selected to give some variety. In the first part of the game the program play is dull but steady. As usual, the program sometimes does not make the best move. About four moves are wasted in this way. The black moves are selected to give the program a maximum amount of trouble. The starting position p_1 is the one given in Capablanca [1935].[4] Capablanca only uses nine white moves for the first part (compared with 21 program moves); however his black king moves are more cooperative than the ones selected in this example.

Our next example is taken from Fine (Figure 6.1). We start from p_2 in Figure 6.15 which is the same as the starting position in Figure 6.1 after adjustments have been made for the fact that the program has the queen's rather than the king's bishop. Again we start in stage 2. We have

1.	wk-QB4	bk-K5
2.	qb-Q6	bk-KB4

4. Page 109.

3.	wk-Q4		Now $m_2=0$ but actually we are in stage 3 with an area of size 6.
		bk-KKt5	This move gives a p with <u>poss</u>(p) true.
4.	wk-K4		White blocks the escape, so the position is accepted by <u>better</u>.
		bk-KKt4	
5.	kt-QB7	bk-KR3	
6.	kt-Q5	bk-KKt3	
7.	kt-QKt6		Now v=3 .
		bk-KB2	
8.	qb-K5		Now v=5 .
		bk-KKt3	
9.	wk-KB4	bk-KR3	
10.	wk-KB5	bk-KR2	
11.	qb-KB6		Now we are in an area of size 4.
		bk-KKt1	
12.	kt-QB8	bk-KR2	
13.	kt-Q6		Now v=5 .
		bk-KR3	
14.	kt-K4		This is the first allowable knight move.
		bk-KR4	
15.	kt-KKt5		This is the second knight move. We have <u>sl</u>(q)=15 .
		bk-KR5	
16.	qb-K5		Now we are in stage 5, and $m_5=4$.

There is no point in continuing the example since the program will play the same as in example 1. Ten more moves are required to mate. As expected, the program plays differently from Fine. The moves for black are chosen to illustrate how the program reaches stage 5 through areas of size 6 and 4. When this path is chosen, stage 5 is short and checkmate is reached quickly.

We will now give two short examples to illustrate special cases in the first part of the game. The next example shows how the program handles a temporary escape from an area. We begin at position p_4 in Figure 6.12. $\underline{st}(p_4)=3$, $s(p_4)=5$, and $\underline{poss}(p_4)$ is true. We have

1.	wk-Q5	bk-QKt5	
2.	kt-K4	bk-QKt6	
3.	wk-Q4	bk-QB7	
4.	qb-QKt4		Now $\underline{sl}(q)=4$.
		bk-Q8	
5.	wk-Q3	bk-QB8	
6.	qb-QB3		Now we are in an area of size 4.

If at any time the black king had returned to the area of size 5, he would have been trapped there and that branch would have terminated.

Our final example shows what happens when we must cope with a stage 2 position complicated by the locations. We start at p_3 in Figure 6.15. p_3 is in stage 2 but all of its immediate successors are in stage 0 or stage 1.

1. kt-KB5		wk-KKt6 would give $m_2(q)<m_2(p)$ and $d_q(wk,qb)\leq d_q(bk,qb)$ but this position is correctly recognized as a member of stage 0. We have q in stage 1.
	bk-KKt1	
2. qb-KB6		We are in stage 2, but $m_2(q)=m_2(p)$. Note $d_q(wk,qb)=2$.
	bk-KB2	
3. wk-KKt5		Now we can accept q as <u>better</u> since $m_2(q)<m_2(p)$.
	bk-K3	
4. kt-KKt7 ch.		The knight was blocking the path of the king.
	bk-Q3	
5. wk-KB5		Now $m_2(q)=1$.
	bk-Q4	
6. wk-KB4		The black king is blocking the white king move into the center.
	bk-QB4	
7. wk-K4		Now $m_2(q)=0$.
	bk-QB5	
8. qb-Q4		Now we are in stage 3.

The program manages nicely.

This last example indicates that the program should be able to reach checkmate from any starting position within the 50 move limit. Stages 5 and 6 together never require more than 19 moves, and the first example of program play gives a close to maximum number of moves through stage 3. Since this example ends similarly to the first example after

move 6, this means the program still has a margin of 8 moves to take care of any complications which arise.

The remarks about the previous end games are also valid here. However, the mediocre (better but not best) program moves are not so frequent in this game. This is because the difficulty of winning forces more exactness in program play. The difficulty of this game also provides a good test of the program. The fact that the program can win, using the fairly simple patterns which provide the outline of the play, indicates that the forcing tree model used for the program is a good one. Also the program play is identical to the book's when sufficient information is available.

BLANK PAGE

CHAPTER 7

PROGRAM CORRECTNESS

Now that the definitions of *better* and *worse* have been given for the various end games, we can consider the question of program correctness. We will say that the program plays an end game correctly if we can prove that it will reach checkmate from any legal starting position $p \in P$. To prove, given the position $p \in P$, that the program will actually win from p, we must show

1. The program can force positions q which are *better* than p.
2. This process need only be repeated a finite number of times before checkmate is reached.

First we must introduce some notation.

Defn. $\underline{prog_1}(p) = \{q \mid q$ is at the end of a branch of the tree from p which is produced by the program$\}$.

If an immediate successor q of p is *better* than p, then $\underline{prog_1}(p)$ will contain the single element q. If the program is unable to force *better* positions from p, we would have $\underline{prog_1}(p)=NIL$, which means either that all branches are rejected or that the program does not terminate (in 50 moves). The first statement can therefore be written:

Theorem 1. $\forall p[p \in P \supset \neg\, null(\underline{prog_1}(p))]$.

Proof. This theorem must be proved separately for the different stages and measures within each end game. It is sufficient to show that an

acceptable path exists; we will not know for certain what $\underline{prog}_1(p)$ contains but we will know that it is not empty since the program uses a breadth first search.

We give a proof here for positions in stage 2 of the Rook end game. Recall that stage 2 is defined by

$$x \in \text{stage } 2 \equiv \{\underline{goodquad}(x) \wedge \underline{squad}(x) > 2\} .$$

The measure in stage 2 is

$$m_2(x) = \underline{squad}(x) \qquad \forall x(x \in \text{stage } 2) .$$

<u>better</u> for stage 2 is defined by

$$\{\underline{st}(p)=2 \wedge (\underline{st}(q)>2 \vee [\underline{st}(q)=\underline{st}(p) \wedge (m_2(q)<m_2(p) \vee d_q(wk,r)<d_p(wk,r))])\} ,$$

and <u>worse</u> by

$$\{\underline{st}(p)=2 \wedge ([\underline{st}(q)=2 \wedge m_2(q)>m_2(p)] \vee [(\underline{st}(q)=1 \vee [\underline{st}(q)=2 \wedge m_2(q)=m_2(p)]) \wedge d_p(wk,r)=1 \wedge (d_q(wk,r)>1 \vee [\underline{st}(q)=1 \wedge r_p \neq r_q])])\} .$$

We divide the proof into two parts depending on $d_p(wk,r)$.

1. $d_p(wk,r)>1$. Then there exists a q with $d_q(wk,r)<d_p(wk,r)$. q will have the same quadrant as p , and since p satisfies $d_p(wk,r) \leq d_p(bk,r)+1$, we can be sure that $d_q(wk,r) \leq d_q(bk,r)$. This q will be <u>better</u> than p , and $\underline{prog}_1(p)=\{q^*\}$. (q^* is not necessarily equal to q.)

2. $d_p(wk,r)=1$. There are two cases to consider. Let $p'=p$ or a successor of some q down in the tree from p .

   a. There is a rook move leading to a position q (in stage 2 or 3) with a smaller quadrant. Such a position q will be <u>better</u> than p , and so we know the tree terminates. We are always in case 2a

if $d_{p'}(bk,r)>2$.

b. No such rook move exists.

i. $d_{p'}(bk,r)=1$ and we are not in 2a. Then we make one of the king moves such that $d_q(wk,r)=1$ and $\underline{fr}_q(bk,wk)<4$ for $\underline{fr}$ as defined in Appendix A. A move like this always exists and is not worse; $\underline{fr}_q(wk,bk)<4$ insures that after the black king moves we will be in 2a or 2bii, which means the tree will terminate in one or two more moves.

ii. $d_{p'}(bk,r)=2 \wedge \underline{fr}_{p'}(bk,wk)=4$ and we are not in 2a. This is the place where the white king moves onto the boundary of the quadrant. Then after the black king moves we are in case 2a with just one move to terminate the tree.

iii. $d_{p'}(bk,r)=2 \wedge \underline{fr}_{p'}(bk,wk)=5$ and we are not in 2a. We make a white king move such that $d_q(wk,r)=1$, and after the black king moves we are in case 2bi or 2a (at most three more moves to terminate the search).

Obviously such proofs are very tedious and we will not attempt to give them for the other stages. The method of proof remains the same, and sketches of such proof have been given in the various chapters.

Although the example chosen for the proof of the previous theorem was given using the practical definition of better, for the rest of this discussion we will use the formal definition of better. We will discuss the extension of the theorems to the practical definition after they have been proved.

First we must prove that rule 3.7 holds.

Theorem 2. $\forall q\ \forall p(qM_B p \supset [\underline{st}(p) \rhd \underline{st}(q) \vee (\underline{st}(p)=\underline{st}(q) \wedge m_{\underline{st}(p)}(p) \leq m_{\underline{st}(p)}(q))])$.

Proof. Again we must prove this for the different stages and measures. In fact we have proved it informally in the chapters covering the end games. The reason it is possible to prove this is that stages and measures depend almost entirely upon the position of the white pieces. When a rule is made about the position of the black king it is stated in q and in p in such a way that if it holds in q, it will hold in all immediate successors p of q.

We have purposely given informal proofs for Theorems 1 and 2 because the detail required for a formal proof is excessive and uninstructive. It is necessary in these theorems to give separate proofs for each stage of each end game. The proof given for Theorem 1 is correct for stage 2 of the Rook end game, and serves as an example of how such proofs should proceed, both for Theorem 1 and Theorem 2, although the proofs for Theorem 2 are simpler.

$\underline{prog}_1(p)$ produces only one step of the program. To handle the entire program we make the following definition

Defn. For $i>1$,

$$\underline{prog}_i(p)=\{q \mid \exists p'q'(q' \in \underline{prog}_{i-1}(p) \wedge q'M_B p' \wedge q \in \underline{prog}_1(p'))\} .$$

Please note that the i in $\underline{prog}_i(p)$ does not generally stand for the ith move from p; it stands for the ith iteration of the program. A new iteration is not begun until the tree (possibly of depth 1) from the previous program entry is exhausted.

Now we can formalize the second statement.

Theorem 3. $\forall p[p\in P \supset \exists K\ \forall N(N\geq K \supset \mathrm{null}(\underline{prog}_N(p)))]$.

Proof. P is associated with some end game, and let us suppose this game has n stages. For each stage i , let k_i be the number of different values which the measure m_i assumes. We know $k_i \geq 1$ for all i . (k_i must be finite; this is true for all the measures which have been defined.) Let

$$K = \sum_{i=1}^{n} (k_i) .$$

K is the number of different categories into which positions in the end game can be put, not counting stage 0. We refer to each category as a level, and we define a function $\underline{le}$, which gives the level of a position as follows.

(1) $\underline{le}(x)=1 \equiv [\underline{st}(x)=1 \wedge m_1(x)=\max_{y\in S}(m_1(y))]$, for $S=\{y \mid \underline{st}(y)=1\}$.

(2) Assume we have defined the set of positions x for which $\underline{le}(x)=i$. If this set is empty, then so is level i+1 . Otherwise, we define the set for which $\underline{le}(x)=i+1$ as follows. Let x be a position such that $\underline{le}(x)=i$.

If i=K , then x is a checkmate position and the i+1 level is empty.

Otherwise $\underline{st}(x)<n$. If $m_{\underline{st}(x)}(x)>\min_{y\in S}(m_{\underline{st}(y)}(y))$, for $S = \{y \mid \underline{st}(y)=\underline{st}(x)\}$, we have

$$\underline{le}(z)=i+1 \equiv [\underline{st}(z)=\underline{st}(x) \wedge m_{\underline{st}(x)}(z)=\max_{y\in S}(m_{\underline{st}(x)}(y))] ,$$

$$\text{for } S = \{y \mid \underline{st}(y)=\underline{st}(x) \wedge m_{\underline{st}(x)}(y)<m_{\underline{st}(x)}(x)\} .$$

Otherwise we have

$$\underline{le}(z)=i+1 \equiv [\underline{st}(z)=\underline{st}(x)+1 \wedge m_{\underline{st}(z)}(z)=\max_{y\in S}(m_{\underline{st}(z)}(y))] ,$$

$$\text{for } S = \{y \mid \underline{st}(y)=\underline{st}(z)\} .$$

For completeness we define

$$\underline{le}(x)=0 \equiv \underline{st}(x)=0 .$$

The levels have the same order as we would like the program to follow; we know $0 \leq \underline{le}(x) \leq K$, for all $x\in Q$. Recall $Q=P \cup \{q \mid \exists p(p\in P \wedge pM_W q)\}$. We have the following lemmas.

Lemma 1. $\forall pq[(q\in Q \wedge qM_B p) \supset \underline{le}(p) \geq \underline{le}(q)]$.

Proof. This follows immediately from Theorem 2.

Lemma 2. $\forall pq[(p\in P \wedge q\in \underline{prog}_1(p)) \supset \underline{le}(q) > \underline{le}(p)]$.

Proof. Since $q\in \underline{prog}_1(p)$, we know $\underline{better}(p,q)$ is true. Therefore, $\underline{le}(q) > \underline{le}(p)$.

Now, for $p\in P$ and $N \geq K$, let us assume there exists a $q\in \underline{prog}_N(p)$. We unravel the meaning of this:

$$q\in \underline{prog}_N(p)$$

$$\exists p_1 q_1 (q_1 \in \underline{prog}_{N-1}(p) \wedge q_1 M_B p_1 \wedge q\in \underline{prog}_1(p_1))$$

$$\vdots$$

$$\exists p_1 q_1 \ldots p_{N-1} q_{N-1} (q_{N-1} \in \underline{prog}_1(p) \wedge q_{N-1} M_B p_{N-1} \wedge \ldots \wedge q\in \underline{prog}_1(p_1)) .$$

We select the appropriate $p_1, q_1, \ldots, p_{N-1}, q_{N-1}$, and apply our lemmas to get

$$\underline{le}(q_{N-1}) > \underline{le}(p) \wedge \underline{le}(p_{N-1}) \geq \underline{le}(q_{N-1}) \wedge \ldots \wedge \underline{le}(q) > \underline{le}(p_1) .$$

Each time we have $\underline{le}(q_i) > \underline{le}(p_{i+1})$ we can write $\underline{le}(q_i) \geq \underline{le}(p_{i+1})+1$ since $\underline{le}$ is an integer function. So we have

$$\underline{le}(q_{N-1}) \geq \underline{le}(p)+1 \wedge \underline{le}(p_{N-1}) \geq \underline{le}(q_{N-1})$$

$$\underline{le}(q_{N-2}) \geq \underline{le}(p_{N-1})+1 \wedge \underline{le}(p_{N-2}) \geq \underline{le}(q_{N-2})$$

Therefore $\underline{le}(q_{N-2}) \geq \underline{le}(p)+2 \wedge \underline{le}(p_{N-2}) \geq \underline{le}(p)+2$.

$$\vdots$$

$$\underline{le}(q_1) \geq \underline{le}(p_2)+1 \wedge \underline{le}(p_1) \geq \underline{le}(q_1) .$$

This gives $\underline{le}(p_1) \geq \underline{le}(p)+N-1$ and since $\underline{le}(q) \geq \underline{le}(p_1)+1$, we have

$$\underline{le}(q) \geq \underline{le}(p)+N \geq \underline{le}(p)+K \geq K+1 ,$$

but this is impossible since $\underline{le}(x) \leq K$ for all $x \in Q$. Therefore $\underline{prog}_N(p)$ is empty.

Theorem 3 insures that the program will never get into a loop. It says that k_i , in addition to being the number of values the measure m_i assumes in stage i, is also a bound on the number of times the program can produce *better* positions in stage i as it moves along from a starting position to checkmate. The proof of Theorem 3 depends entirely upon Theorem 2 and the definition of *better* (Lemmas 1 and 2).

We use this theorem as follows. Consider how the set $\underline{prog}_N(p)$ is formed. There are two parts to the definition. One part looks like $q_i \in \underline{prog}_1(p_{i+1})$; the other is a statement like $q_i M_B p_i$. Now Theorem 1 says that the statement $q_i \in \underline{prog}(p_{i+1})$ is always true provided $p_{i+1} \in P$. We know this for the original p . However we must show

Theorem 4. $\forall p^*[\exists pq(p\in P \wedge q\in \underline{prog}_1(p) \wedge qM_B p^*) \supset p^*\in P]$.

Proof. This proof is the same for all end games and it produces a condition on the definition of stages. If we assume the premise for some p^* , then we know $\underline{le}(p^*)\geq 2$, since $\underline{le}(p)\geq 1$. This means that all non-winning positions which can be produced from a winning position must be below the second level. In all three of the games discussed the second level is in stage 2. The only questionable game is the Bishop-Knight; we are confident that there is no $p\in P$, $\underline{st}(p)=1$, which produces p^* , $\underline{st}(p^*)=2$, but $p^*\notin P$, in this end game.

By Theorems 1 and 4 we can be sure that the chain leading to $\underline{prog}_N(p)$ does not fail because a set $\underline{prog}_1(p_{i+1})$ is empty. Therefore it must fail in the other statement , $q_i M_B p_i$. This can only happen if some q_i has no successors. But if q_i has no successors it is either stalemate or checkmate. In this case it cannot be stalemate since we know it is <u>better</u> than some p ; therefore it must be checkmate. So Theorem 3 means that less than K uses of $\underline{prog}_1$ are required to reach checkmate for any $p\in P$. Therefore we can say

Theorem. $\forall p(p\in P \supset$ the program will force checkmate from p).

Before leaving the subject of correctness we must discuss the extension of these theorems to the practical definitions of <u>better</u> and <u>worse</u>. Theorem 4 is the only one which is unaffected by the additions. We consider Theorems 1, 2 and 3.

We first realize that Theorem 1 is not affected by the additions to <u>better</u>. This theorem is really a statement of existence and if the

program terminates sooner than expected this does not affect the proof. Theorem 1 is affected, however, by the additions to worse. We must be sure that worse does not now eliminate the path which is followed for the proof of Theorem 1. When we used the formal definitions of better and worse there was no danger of this sort because worse hardly eliminated anything. Recall the formal definition of worse

$$\underline{\text{worse}}(p,q) \equiv [\underline{\text{st}}(q)=0 \vee (\underline{\text{st}}(p)=\underline{\text{st}}(q) \wedge m_{\underline{\text{st}}(p)}(p)<m_{\underline{\text{st}}(p)}(q))] .$$

Since all positions in stage 0 were disastrous, only the second part of the rule could affect the eventual finding of better positions. This problem was considered carefully as the stages and measures were defined for each end game, and only if we were sure the program would work correctly was a function allowed to be a measure. The proof of Theorem 1 is based upon this fact. Similar care must be exercised when additions are made to worse. This problem is considered in Chapters 4, 5 and 6, when the additions to worse are described.

Theorem 2 is the statement of rule 3.7, and must be extended to cover each addition to better. This extension was discussed as the additions were made, but we will consider it again here. Theorem 3 is affected by the additions because we must redefine K. We discuss both theorems at the same time.

In the Rook end game only one addition is made to the formal definition of better; this is

$$\underline{\text{st}}(p)=\underline{\text{st}}(q)=2 \wedge m_2(p)=m_2(q) \wedge d_q(wk,r)<d_p(wk,r) .$$

As was mentioned in Chapter 4, this use of d is like a measure. Since only the position of white pieces is involved we can be sure that the evaluation of a successor of q using d will give the same

value as q ; therefore the correctness of this addition depends on the correctness of stage 2 and its measure. In Theorem 3, we must use a different value for K by replacing k_2 with k_2^* , where $k_2^*=7\cdot k_2$, since $d_x(wk,r)$ can have at most 7 different values.

In the Bishop-Knight end game we need only worry about stage 3. As was mentioned in Chapter 6, s satisfies 3.7, and $k_3=3$. v also is nearly a measure and 3.7 can be extended to cover it since it depends only on the position of the white pieces. v leads us to give a value of $k_3^*=9$.

In the Two-Bishops end game, we added a function which is not like a measure since it is not integer-valued. This is the rule used for non-head quadrants in stage 2:

$$\underline{st}(p)=\underline{st}(q)=2 \wedge \underline{squad}(p)=\underline{squad}(q) \wedge (\underline{squad}(p) \text{ is even})$$
$$\wedge\, d_q(wk,bk)<d_p(wk,bk) \wedge \underline{dmin}(q)<\underline{dmin}(p) .$$

This rule is acceptable because of the use of $\underline{dmin}$ which is a function of white pieces only. Therefore we know that

$$qM_Bp \supset \underline{dmin}(p)=\underline{dmin}(q) .$$

Also the rule can be applied no more than six times since $\underline{dmin}(x)\leq 6$ for all x with $\underline{st}(x)=2$. $\underline{dmin}$ could be used as a measure by itself. We can think of the other part of the rule,

$$(\underline{squad}(p) \text{ is even}) \wedge d_q(wk,bk)<d_p(wk,bk)$$

as a modifier on $\underline{dmin}$. It does not affect the extension of Theorems 2 and 3.

CHAPTER 8

EVALUATIONS AND CONCLUSIONS

Evaluation of the Forcing Tree Model

We concider first the forcing tree model selected for the program. This model has proved to be a good one for our purposes. The end games described have all led to fairly simple pattern descriptions. Also, we have been able to prove that the program can reach checkmate from a given starting position. This proof depends heavily on the model, which is represented by functions $\underline{prog}_1$ and $\underline{prog}_n$.

As far as the quality of program play is concerned, the program plays all of the end games discussed in quite a reasonable manner. The main objection which can be made is that the program does not always play as well as it might. Sometimes when there is a perfectly obvious move which produces a position much better than the present one, the program will select another move which is not as good.

Such play is a natural consequence of a method which looks for a good move rather than the best move. And obviously, the more heuristics the program has the more likely it is that the best move will not be selected. For example if only checkmate positions were recognized by better the best move would always be selected. However this approach is not practical because the tree search is too large. In general there is this trade-off between goodness of play and length of tree search.

There are several fairly simple ways of making the program play more efficiently. First of all we could improve program play by having it search for the best move, rather than just settling for a good one. This is easy to implement when examining immediate successors q of some position p. We would simply let Q be the list containing all q which were *better* and not *worse* than p. Then after all successors of p had been examined, if Q were not empty we would compare the members of Q with each other, using a function similar to the formal definition of *better*. The formal definition could not be used because it expects a position with white to move as its first argument. However function

8.1 $\underline{\text{better}}_Q(q,q') \equiv (\underline{st}(q') > \underline{st}(q) \vee m_{\underline{st}(q)}(q') < m_{\underline{st}(q)}(q))$

could easily be defined to compare two positions with black to move.

We convert the formal definition of *better* rather than the actual one for two reasons. First of all, there is so little difference between two positions, both successors of the same position p with the same stage and measure, that it is not worth the extra work to distinguish between them. However even if we wanted to, it is not always possible to convert the actual value of *better* into a rule like 8.1, because sometimes some information about p is used to assign a value to q in this definition. For example, in Two-Bishops we use the fact that p is non-head quadrant to decide about q. This decision really depends on the fact that p is a predecessor of q, and cannot be converted into a comparison of two positions with black to move.

It is not simple to extend this method of program improvement to tree search because the choice of one branch over another is not so

clear-cut. In a tree search it is not usually a matter of deciding which particular q to put at the end of the branch, although this would improve the program somewhat. It is more important to decide between several branches all of which terminate at the same depth. For example, suppose one branch of a tree almost always leads to a much better position than the original p , except in one or two places which are only slightly better, while another branch is neither as good nor as poor as the first. It is difficult to say which branch should be chosen.

The main thing wrong with this method, even if we do not consider the problem of choosing between branches of a tree, is the fact that it would greatly increase program run time. After all, the killer heuristic, discussed in Chapter 2, introduces playing inefficiency but is used because the time saved is more important. This method of searching for the best move would waste more time than is saved by the killer heuristic (and also it is incompatible with the killer heuristic).

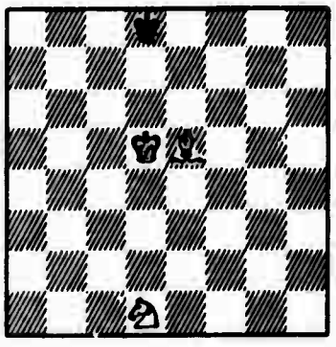
A way of improving program play which is not so time consuming is the following, which compensates for the inefficiency in play introduced by using extra heuristics to avoid tree search. We could replace better with a hierarchy of functions which will be referred to as versions of better. For example, version 1 would recognize gross differences between p and q (for instance, only changes in stage); version 2 would recognize smaller differences and so on. Then all non-worse successors q of some position p would be examined using version 1 of better; if none were selected they would be examined by

version 2; and so on. This would be faster than the previous method because the tests in each version of better would be very short, and as soon as a q was selected, all testing would stop. A gain in efficiency would be made even if just two versions were used; one would be the formal definition of better while the other would be the additions which make the program practical. However three versions would be required to get the most out of the method, because we would always prefer a change in stage to a change in measure.

Another way in which program efficiency could be improved would be by paying attention to the order of move generation. This has already been done to some extent; for example in the Rook end game, rook moves are examined before king moves, so that a smaller quadrant will be formed if possible. On the other hand, in the Bishop-Knight game, knight moves are examined first so that for example in position p_1 in Figure 8.1, kt-KB2 will be selected ($s(q)=6$, $v(q)=5$) , although q-Q6 would give $s(q)=5$. Even if the only ordering done is to decide what piece's moves to examine first, some gain in efficiency can be obtained. More gain in efficiency can be made by considering the ordering of moves for each piece. For example if the rook moves farthest away from the rook were generated first, then in p_2 in Figure 8.1, we would select either r-K5 or r-QB3 giving a quadrant of size 10 or 12. If moves are generated in the opposite way, r-Q5 or r-QB4 would be selected giving a quadrant of size 15 or 16.

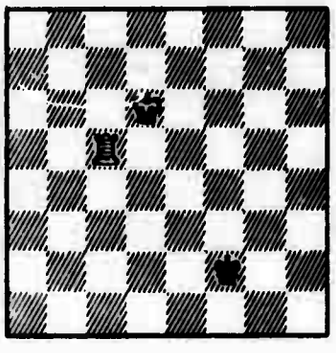
Improving program play by changing the move ordering does not increase the playing time (provided the killer heuristic is allowed to

p_1



p_1 is in stage 3.
$s(p_1) = 6$, $v(p_1) = 1$.
kt–KB2 gives $s(q) = 6$, $v(q) = 5$.
qb–Q6 gives $s(q) = 5$, $v(q) = 1$.

p_2



p_2 is in stage 2, and
$m_2(p_2) = 20$. If r–QB3, then
$m_2(q) = 10$. If r–K5, then
$m_2(q) = 12$. If r–Q5, then $m_q(q) = 16$.
If r–QB4, then $m_2(q) = 15$.

Figure 8.1.

stand). However, many times the move ordering will be wrong for the particular situation. The board is symmetric in many ways in these games, and so it is often possible to think of two positions p which require opposite move ordering if the best position q is to be generated first. Position p_2 in Figure 8.1 is an example. Although we can order the moves so that a quadrant of size 10 or 12 will be selected instead of one of size 15 or 16, there is no way to order the moves so that we can be sure that the quadrant of size 10 will be selected in both p_2 and all positions which are equivalent to p_2 with respect to the symmetry of the board.

Correspondence of Program and Book Methods

Now we consider how closely the definitions of better and worse correspond to the methods described in the chess books. When the information in the books is reasonably complete, we would like the program to play similarly to the books. We feel this goal has been achieved. The only place where the information about play is very inadequate is the first part of the Bishop-Knight game (actually stage 3). In all other parts the information is adequate, and generally there is no question that the program plays the same as the books.

The one exception is stage 2 of the Two-Bishops game. The lack of correspondence here comes from the fact that sometimes the book chooses a move by a different heuristic even though the stage has not changed. The reason the book does this is probably to show the student that more than one kind of method can be applied. In other stages and other games, the number of moves chosen by a different heuristic is very small and so

does not worry us. It is perfectly reasonable to limit the program to one kind of heuristic for each stage, and this is what is done in Two-Bishops.

Another kind of difference between program and book moves is that sometimes the book looks ahead one move (or more) even though it could make a decision immediately. There is no reason to attempt to model this. It does not happen consistently, and does not indicate any essential change in methods.

Evaluation of the Translation Process

We have shown that the forcing tree model allows the program to produce winning play for three end games, one of which is very difficult. Further there is a fairly close correspondence between book and program play. We take this as proof that the model is a good representation of the abstract model assumed by chess players. Now we turn our attention to the difficulty encountered in translating from the books into the definitions of better and worse.

An examination of Chapters 4, 5 and 6 will suffice to convince us that this translation process is surprisingly difficult. Sometimes we are hampered by a lack of book information, but even when there is plenty of information we still encounter difficulty. The reason for this is that the induction required of the student is more extensive than we expected. For example, in the last part of the Bishop-Knight game (stage 5) the chess books give an almost complete example of play. However it is very difficult to decide which features should be used to represent the pattern.

Now if we divide the translation process into simple versus difficult tasks we find the following. It is simple to decide roughly what the stages are, and what kind of heuristic each requires. This information is often stated in the books. It is difficult to give the exact definition of the stages and measures, and generally it is even more difficult to define the additions to better and worse which make them practical. So we ask the question: can we use the computer to help with the translation?

One way in which the amount of work might be lessened is the following, which helps with some of the difficult tasks. First we observe that all the heuristics used in better and worse consist of complicated predicates built up out of simpler predicates joined by propositional calculus connectives. Many of the simpler predicates are useful in all the different games, for example functions d and de. Others are not so widespread but are still basic to the structure of the end game; for example function location in the Bishop-Knight game is a natural function for measuring distances from diagonals.

Next we observe that defining the heuristics for an end game is done in two separate parts. First we give the definitions of the stages and measures, which are taken from the chess books whenever possible. When the game is well defined the process of arriving at the stages and measures, while sometimes tedious, can be guided by the books.

After the formal definitions of better and worse are complete we turn our attention to the practicality of the method. At this point the chess books are not so useful; painstaking examination of the paths

which the program should and should not follow is the important thing. The rules arrived at are built up out of the distance functions and pieces of the definitions of stages and measures. So it is entirely feasible that this part of the definition of heuristics can be done by the program.

The following method assumes that stages and measures have been defined. The program has available to it the definitions and can get at parts of them. It can generate many other functions, in particular the distance functions, and also tests like $wk_p = wk_q$. Whenver it has to do a tree search at some p, then for all q at the top of a branch of the tree, it generates a description of q which is the conjunction of the values of all the functions it has at its disposal. When the tree search terminates it notes which pattern describes the successful branch and which patterns describe branches which failed. Then the next time it encounters a position p' like p it will accept a successor q' of p' which fits a previously successful pattern. If it still has to do a tree search, it will reject all successors of p' which fit a failure pattern. Since q' may actually be in a lower stage than p', the program must remember, when it accepts q' in this way, to use p' as the first argument of <u>better</u> (rather than p'' such that $q' M_B p''$) until it finally reaches a q which is accepted by <u>better</u>. A flow chart of this process is given in Figure 8.2.

The tree search required to implement this method will be very lengthy at first, but will decrease in time. The more simple functions the program has to work with, the longer it will take to converge on a useful pattern. On the other hand, if the program has too few simple

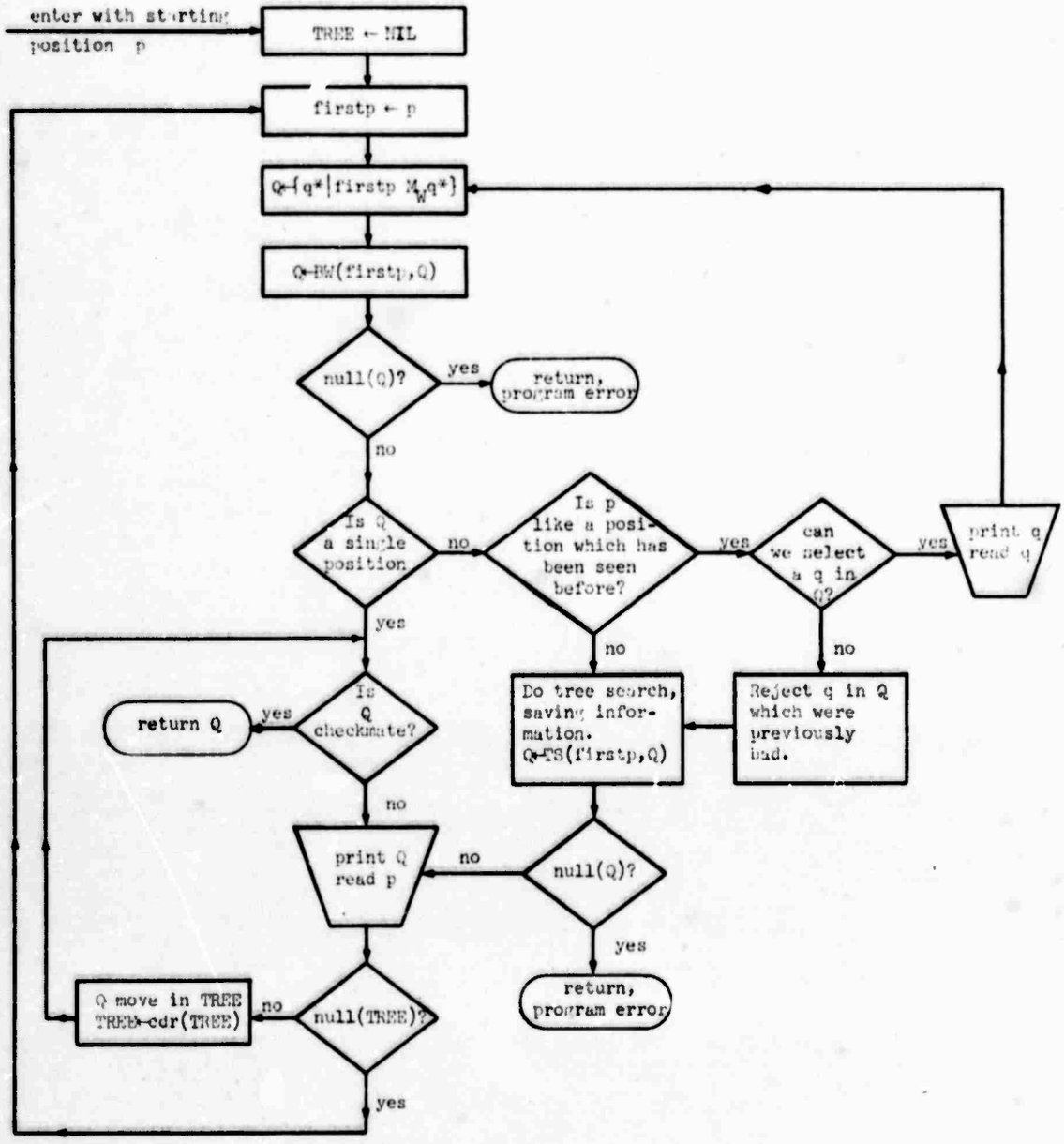


Figure 8.2. Program Organization for Doing Simple Learning.

functions it may make errors in the sense of being unable to distinguish between two positions q and q' , one of which is at the head of a shortest branch, while the other is rejected as leading to worse positions.

This method has been implemented for the Rook end game in an abridged form. The program was given the formal definitions of better and worse, plus the following functions (2-valued or 3-valued; f : g has the 3 values < , = , and >):

quad(q)

squad(p) = squad(q) (If ¬ quad(p) ∨ ¬ quad(q) , the value is undefined. This function is useful only in stage 3, since squad is a measure in stage 2.)

$d_p(wk,r) : d_q(wk,r)$

$d_p(wk,bk) : d_q(wk,bk)$

$d_p(bk,r) : d_q(bk,r)$

$\underline{fr}_p(wk,r) : \underline{fr}_q(wk,r)$

$\underline{fr}_p(wk,bk) : \underline{fr}_q(wk,bk)$

$d_q(wk,r) : d_q(bk,r)$.

These functions were given to it; it did not derive them. In addition the program needed a way of classifying positions p so it would know when its new functions should be used. To classify p , the program used

st(p) (only stages 1, 2, 3 apply)

$d_p(wk,r)=1$.

Thus the program had a maximum of six classifications.

The program used a "complete" tree search, which means that when it discovered a good branch of the tree at some depth n , it continued the tree search for the rest of depth n to see if any other branches were also good at that depth. The program was run on a series of 34 positions requiring tree search; it was able to make moves immediately for 16 of them, including 7 out of the last 10.

The moves which the program discovered were not always the same as the moves which the practical version of <u>better</u> would find. For example it learned to make the moves satisfying

$$d_p(wk,r)>1 \wedge d_q(wk,r)<d_p(wk,r) ,$$

but it also learned to move the white king onto the boundary of the quadrant so that a smaller quadrant could be formed on the next move. Theoretically, of course, it should be possible to make every move without tree search. In ract, this method is quite good at extending the definitions of <u>better</u> but does not develop much of a definition of <u>worse</u>. Note also that this method produces a hierarchy of versions of <u>better</u>, as was discussed in the first part of this chapter, so program play remains quite efficient.

Extensions in Chess

To illustrate the fact that the program is useful, we discuss how it could be extended to cover a larger set of end games. Up to now, chess programs have not been concerned with these games. The program of Baylor and Simon [1966] could not be used to play chess end games. It deals with mating combinations; these are the chess problems, in

which there are many pieces on the board and only a few moves to checkmate. The more general programs (for example, Greenblatt's program [Greenblatt and Crocker, 1967]) are written to play the middle game. The end games are ignored since they are played differently from both the middle game and each other. Therefore, if enough end games could be handled by the program, it would be a useful addition to a more general program like Greenblatt's.

The success of the program has convinced us that it can be applied to other end games. All that is required is the conviction of chess players that the particular end game can be won from all but certain defined positions. If a position truly can be won, this means there must be features of the position which express this fact. The notion of better, using stages and measures, provides a good framework for gathering and using these features.

There are two main problems to consider when extending the program. First, it would seem that the method is not suitable to games in which black has many moves. If black has a few pawns that is all right, but as soon as black has a major piece, there would be too many black moves to do the tree search which the method requires. However, the number of moves is more apparent than real because usually most black moves would be disastrous. One way to take advantage of this fact would be to modify the program to evaluate positions in the tree search after black moves as well as after white moves. After black moves, the program would look for positions which lead to a better position in one white move (there is no reason why this could not be recognized at this point). After white moves possibly only worse would be used,

since we would know better could not be satisfied or we would not have searched so far. With this change, which is not a major one, many games would become amenable to the method; for example King and Queen against King and Rook.

A more serious problem is the fact that the heuristics are different for each end game and this means that better and worse must be redefined. However, if the induction method described in the previous section could be used, we could extend the program without too much difficulty to other end games. For example we could easily include King and Queen against King and the various Pawn end games.

In addition, we can use the program to handle other games without giving new definitions of better and worse. These are end games which include some solved end game as a subset. For example, suppose we had essentially the Bishop-Knight end game, but black had a pawn and white an extra bishop. Then the program could afford to sacrifice a bishop to take the pawn. It would recognize this fact by obtaining at the end of a tree search, which should be fairly short, a position q in the Bishop-Knight game such that $\underline{st}(q) \geq 2$.

Therefore the program can be extended to cover a fairly large set of end games. This means that translating from book methods into program heuristics can produce a useful program, at least in this task area.

Conclusions

The principal goal of this research was to study the process of the translation of book problem solving methods into computer program

heuristics. We chose the task area of chess end games for this work. To isolate the translation process, we distinguished between the model which chess books use for these games, and the methods which are applied to particular end games. We decided to represent the model as closely as possible, so that the translation process would be contained mainly in the representation of book methods by program heuristics.

The forcing tree model chosen for the program has proved to be a good representation of the abstract book model. As a consequence of the closeness of the representation, we are able to express the methods in fairly simple patterns, and in addition we can prove that the program will reach checkmate from a given starting position. The value of the proof comes from the condition (rule 3.7) which it forces us to state. The condition gives us a way of evaluating functions proposed for defining <u>better</u> and <u>worse</u>, which is simpler than trying to think only in terms of sequences of moves, and more likely to be correct. This advantage supports our arguments that the program model should be as close as possible to the abstract model assumed in the book. The first hypothesis should therefore be considered when future efforts in translation of book information are made.

Now we turn our attention to the translation process itself. The main result is that we now see how much induction is required. Induction is a form of learning which we would like to understand better. The example in the preceding section of this chapter leads us to believe that the field of chess end games is a good one in which to study induction. It may be possible to develop a program which

will do most of the work of translating, and research can profitably be done in this direction.

We would also like to extend this translation process to other fields of study. A field which presents itself is integration. When integration is taught in a mathematics text, examples are given showing how the rules should be applied. It seems reasonable that inductive learning is going on here; some pattern in the original expression suggests the application of a certain transformation. The learning is probably less involved than in chess end games. In Slagle's [1963] program he has simply done all the work ahead of time. It would be interesting to see what could be done by trying to use the book more directly.

APPENDIX A

DESCRIPTION OF NOTATION AND DEFINITIONS OF BASIC FUNCTIONS

1. The following abbreviations are used to represent pieces.

bk	black king
wk	white king
r	rook
qb	queen's bishop
kb	king's bishop
kt	knight

2. For x a position, and n the name of a piece,

   n_x = the square which piece n occupies in x .

3. Function d(X1,X2) equals the number of king moves required to move a piece from square X1 to square X2 .

4. Function de(X,e) equals the minimum number of king moves required to move a piece from square X to a square on the edge of the board e .

5. Function fr(X1,X2) equals the difference in files between squares X1 and X2 , plus the difference in ranks.

6. $f_x(n1,n2)$ is used as an abbreviation of $f(n1_x,n2_x)$ when the squares containing pieces n1 and n2 are to be selected from the same position x .

BIBLIOGRAPHY

In addition to the chess books referred to in the body of the thesis, several other books are mentioned here which were also found useful.

Baylor, G. W., and Simon, H. A., 1966, A Chess Mating Combinations Program, Proceedings of the AFIPS Spring Joint Computer Conference, Spartan Books, Washington, D. C., 28: 431-447.

Capablanca, J. R., 1935, A Primer of Chess, Harcourt, Brace, New York.

Fine, R., 1944, Chess the Easy Way, David McKay, New York.

Foster, A. W., and Kemp, R. E., 1943, Chess: An Easy Game, David McKay, New York.

Greenblatt, R. D., and Crocker, S. D., 1967, The Greenblatt Chess Program, Proceedings of the AFIPS Fall Joint Computer Conference, Thompson, Washington D. C., 31: 801-810.

Horowitz, I. A., 1957, How to Win in the Chess Endings, David McKay, New York.

Mason, James, 1905, The Art of Chess, Howard Cox, London.

McCarthy, J., Abrams, P. W., Edwards, D. J., Hart, T. P., Levin, M. I., 1965, LISP 1.5 Programmers Manual, The M.I.T. Press, Cambridge, Massachusetts.

Minsky, M. L., 1961, Steps Toward Artificial Intelligence, Proceedings of the I.R.E., 8-30; reprinted in Computers and Thought, Feigenbaum, E., and Feldman, J. (Ed), McGraw-Hill, New York, 406-450.

Newell, A., Shaw, J. C., and Simon, H. A., 1957, Empirical Explorations with the Logic Theory Machine, Proceedings of the 1957 Western Joint Computer Conference, I.R.E., New York, 15: 218-239.

Newell, A., and Simon, H. A., 1961, GPS - A Program That Simulates Human Thought, Lernende Automaten, H. Billing, Munich, 109-124.

Slagle, J. R., 1963, A Heuristic Program That Solves Simple Symbolic Integration Problems in Freshman Calculus, Computers and Thought, Feigenbaum, E., and Feldman, J. (Ed), McGraw-Hill, New York, 191-203.

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
Artificial Intelligence Project
Computer Science Department
Stanford University

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE

A Program to Play Chess End Games

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
A.I. Memo

5. AUTHOR(S) (First name, middle initial, last name)

Barbara J. Huberman

6. REPORT DATE
19 August 1968

7a. TOTAL NO. OF PAGES
168

7b. NO. OF REFS
12

8a. CONTRACT OR GRANT NO.
CS 106

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
AI-65

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

Statement No. 1 - Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
Advanced Research Projects Agency

13. ABSTRACT

A program to play chess end games is described. The model used in the program is very close to the model assumed in chess books. Embedded in the model are two predicates, better and worse, which contain the heuristics of play, different for each end game. The definitions of better and worse were obtained by programmer translation from the chess books.

The program model is shown to be a good one for chess end games by the success achieved for three end games. Also the model enables us to prove that the program can reach checkmate from any starting position. Insights about translation from book problem solving methods into computer program heuristics are discussed; they are obtained by comparing the chess book methods with the definitions of better and worse, and by considering the difficulty encountered by the programmer when doing the translation.

Do not use underlining.

DD FORM 1 NOV 1473

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
better						
worse						
heuristics						
end games						
tree search						
quadrants						